Zeroth-Order Fine-Tuning of LLMs with Transferable Static Sparsity

Anonymous authors
Paper under double-blind review

Abstract

Zeroth-order optimization (ZO) is a memory-efficient strategy for fine-tuning Large Language Models using only forward passes. However, applying ZO fine-tuning in memory-constrained settings such as mobile phones and laptops remains challenging since these settings often involve weight quantization, while ZO requires full-precision perturbation and update. In this study, we address this limitation by combining static sparse ZO fine-tuning with quantization. Our approach transfers a small, static subset (0.1%) of "sensitive" parameters from pre-training to downstream tasks, focusing fine-tuning on this sparse set of parameters. The remaining untuned parameters are quantized, reducing memory demands. Our proposed workflow enables efficient ZO fine-tuning of an Llama2-7B model on a GPU device with less than 8GiB of memory while outperforming full model ZO fine-tuning performance and in-context learning.

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, fine-tuning LLMs for specific tasks remains necessary to achieve optimal results. However, the fine-tuning process often requires significantly more memory compared to inference. Specifically, there are four main components that occupy the memory during fine-tuning LLMs: **(1)** the weight parameter itself; **(2)** the optimizer state, which contains the information about the past gradient (Kingma & Ba, 2015); **(3)** the gradient used to update the parameters; **(4)** the activation cached to calculate the weight gradient (Liu et al., 2024c); Previous work, such as QLoRA (Dettmers et al., 2023), has successfully reduced memory usage for both **(1)** and **(2)** by combining weight quantization and low-rank adaptation (Hu et al., 2021), which enables fine-tuning huge LLMs under consumer level GPUs. However, on memory-constrained hardware like smartphones, the memory required for caching **(3)** gradient and **(4)** activations for backpropagation remains significant. Prior approaches to address this issue are often system-based, such as CPU offloading. The disparity between the memory demands of LLM fine-tuning and hardware capacity limits the adaptability of LLMs, especially when personalizing them for edge devices.

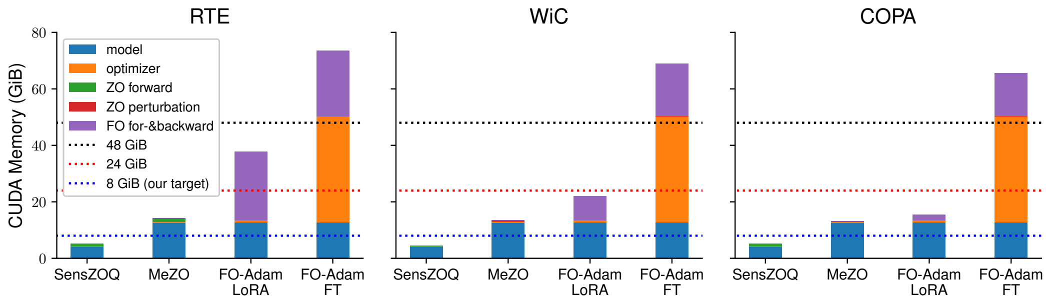


Figure 1: CUDA memory benchmarking of Llama2-7B on 3 fine-tuning tasks. We use a batch size of 8 for profiling the memory usage, and we find that SensZOQ (1st bar in each subfigure) can meet the 8 GiB memory target *without any system-level solutions such as CPU offloading*.

Exploring zeroth-order optimization in LLM fine-tuning. Recently, there has been a resurgence of interest in zeroth-order (ZO) optimization methods for LLM fine-tuning (Malladi et al., 2023a; Liu et al., 2024a; Chen et al., 2024). ZO optimization method perturbs model parameters in random directions and utilize the loss value difference to compute the gradient direction for parameter updates. A key advantage of ZO methods in LLM fine-tuning is that they do not require backpropagation procedures, significantly reducing computation and memory requirements. Being backpropagation-free, ZO methods does not need to cache **(3)** gradients and **(4)** activations during fine-tuning. In practice, ZO methods have demonstrated the potential to achieve performance comparable to first-order methods in LLM fine-tuning, which create new venues for efficient LLM adaptation strategies.

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., 2024b), achieving approximately $\sim 75\%$ *dynamic* sparsity (perturb & tune 25% parameters per step). 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. 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, dynamic sparsity leads to a reduction in training iterations *but not necessarily wall-clock time* – determine and apply the sparsity pattern for each training step could be expensive. Moreover, *dynamic sparsity inherently assumes the whole model must all be in dense weights*, and an attempt to combine dynamic sparse training with parameter-size reduction techniques such as quantization is not computationally tractable (otherwise it will involve frequent dequantization and quantization). This raises the question:

Is it possible to develop an extreme static sparsity method for ZO fine-tuning that is easy to combine with quantization method? Would the memory-efficiency of ZO be even pushed further?

Our proposal: ZO LLM fine-tuning with transferable static sparsity. In this paper, we answer the raised research question by proposing a transferable static sparse ZO LLM fine-tuning strategy. We observe an extreme sparsity pattern in LLM parameters: a subset, determined by selecting the top k magnitude entries from the diagonal of empirical Fisher information matrix, is effective for ZO fine-tuning. Moreover, we find this sparsity pattern can be obtained through LLM's pre-training process and transferred to various downstream tasks without modification (as a static selection).

Summary of contributions. Building on these insights, our work proposes a comprehensive framework for ZO fine-tuning, making the following contributions:

- We identify that only an extremely small portion (**0.1%**) of LLM parameters should be updated during ZO LLM fine-tuning. Moreover, we observe that this sparsity pattern can be derived in LLM pre-training process and transferred across different downstream tasks while still maintaining good ZO performance without any modification.
- Based on this observation, we propose SensZOQ, an on-device LLM personalization workflow via integrating **Sens**itive **ZO** optimization with **Q**uantization to further improve the memory-efficiency of ZO fine-tuning (Figure 1).
- We conduct extensive experiments across various LLMs and demonstrate that our method achieves competitive performance across various downstream tasks.

2 SPARSE ZO FINE-TUNING WITH STATIC SENSITIVE PARAMETERS IN LLM

In this section, we first go through the background of ZO optimization in Section 2.1. We then inspect an extreme sparsity pattern in LLMs in Section 2.2 and its theoretical guarantees in Section 2.3.

2.1 ZEROTH-ORDER OPTIMIZATION

ZO surrogate gradient estimator. ZO optimizers have been studied widely in the machine learning community. 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 will estimate the gradient at $\mathbf{w}$ via ZO surrogate gradient estimator. Simultaneous Perturbation Stochastic Approximation (SPSA) (Spall, 1992) is such an estimator that would first sample a random vector $\mathbf{z} \in \mathbb{R}^d$ and uses the *loss value difference* to scale the update direction. $\mathbf{z}$ is usually sampled from an Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{I}_d)$.

Definition 1 (**Simultaneous Perturbation Stochastic Approximation (SPSA)** (Spall, 1992))**.** *SPSA estimates the gradient w.r.t.* $\mathbf{w}$ *with a data example* $(\mathbf{x}, y)$*, a small constant* $\epsilon \in \mathbb{R}$*, and a sampled random vector* $\mathbf{z} \in \mathbb{R}^d$ *as follows:*

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

There are other ZO surrogate gradient estimators (Liu et al., 2020; Ohta et al., 2020), but in practice SPSA achieves good performance in ZO fine-tuning. Some ZO algorithms such as DeepZero (Chen et al., 2024) would utilize the *parameter-wise* finite difference of loss values to derive *parameter-wise* update directions. This would yield $O(d)$ query costs per training step *even when combining with certain sparse masking methods* and not practical for LLM fine-tuning scenarios. We therefore select SPSA with random Gaussian perturbation as our ZO gradient estimator.

ZO-SGD algorithm. ZO-SGD is an optimizer similar to SGD but replaces the FO gradient with ZO surrogate gradient estimate per training step, as defined below:

Definition 2 (**ZO-SGD update rule**)**.** *ZO-SGD is an optimizer that uses ZO surrogate gradient to update parameters* $\mathbf{w}_t$ *with learning rate* η_t *and a data example* $(\mathbf{x}_t, y_t)$ *sampled at timestep* t*:*

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

MeZO (Malladi et al., 2023a) is a ZO-SGD algorithm that uses the “random seed trick” to save the need of caching ZO surrogate gradient. The choice of optimizer (SGD) is orthogonal to ZO optimization techniques, but in our preliminary experiments we find adaptive optimizers such as Adam (Kingma & Ba, 2015) would not necessarily accelerate ZO convergence in LLM fine-tuning scenarios. There are other ZO optimizers aware of the parameter-wise heterogeneity of loss curvatures to accelerate the optimization convergence (Zhao et al., 2024), and we leave how to combine our method with theirs as future works.

2.2 SPARSE ZO OPTIMIZATION WITH STATIC SENSITIVE PARAMETERS.

Given model parameters $\mathbf{w}$, a loss function f, a data example $(\mathbf{x}, y)$, sensitive parameters are defined as *parameters whose corresponding FO coordinate-wise gradient square values are maximized.*

Definition 3 (**Sensitive parameter mask**)**.** *A sensitive sparse mask* $\mathbf{m}_k \in \{0, 1\}^d$ *with* k *nonzero entries* ($\sum_i \mathbf{m}(i) = k$) *is defined as*[1]

$$\mathbf{m}_k = \mathit{argmax}_{\mathbf{m}} \|\mathbf{m} \odot \nabla f(\mathbf{w})\|_2^2. \tag{3}$$

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 4 (**Sensitive sparse ZO-SGD update rule**)**.**

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

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 ZO SPSA estimator.** The square (account for negativity) of loss value difference for $\hat{g}_{\mathbf{w}}(\mathbf{w}, (\mathbf{x}, y), \bar{\mathbf{z}})$ is as follows:

  $$\mathbb{E}_{\bar{\mathbf{z}}}\{f(\mathbf{w} + \epsilon\bar{\mathbf{z}}; (\mathbf{x}, y)) - f(\mathbf{w} - \epsilon\bar{\mathbf{z}}; (\mathbf{x}, y))\}^2 \approx \mathbb{E}_{\bar{\mathbf{z}}}\{2\epsilon\bar{\mathbf{z}}^\top \nabla_{\mathbf{w}} f(\mathbf{w})\}^2 = 4\epsilon^2 \|\mathbf{m}_k \odot \nabla_{\mathbf{w}} f(\mathbf{w})\|_2^2$$

  Since by Definition 3 our sensitive mask would maximize $\|\mathbf{m}_k \odot \nabla_{\mathbf{w}} f(\mathbf{w})\|^2$ 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*. This property is important for ZO as ZO directly leverages the loss-value difference as a probe of loss landscape to determine the descent direction.

---

[1] When the context is clear, we will abbreviate $f(\mathbf{w}; (\mathbf{x}, y))$ as $f(\mathbf{w})$ and $\nabla f(\mathbf{w}; (\mathbf{x}, y))$ as $\nabla f(\mathbf{w})$. Notice that for full batched gradient we will use $\nabla \mathcal{F}(\mathbf{w})$.

- **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:

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

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.

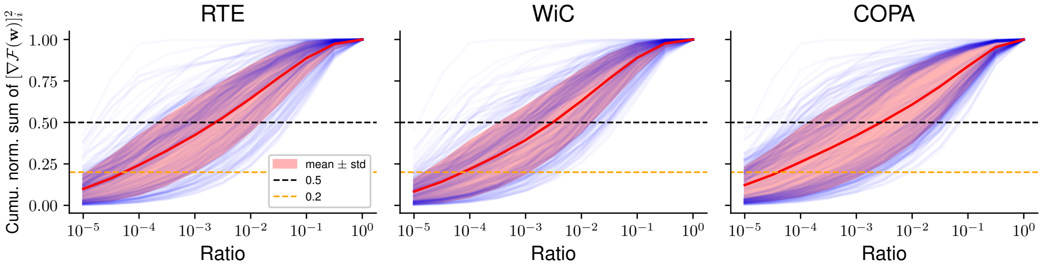


Figure 2: Cumulative normalized sum of coordinate-wise $[\nabla \mathcal{F}(\mathbf{w})]_i^2$ of linear layers during Llama2-7B (Touvron et al., 2023) fine-tuning. For each linear layer, we first sort parameters by the decreasing order of their gradient square value $[\nabla \mathcal{F}(\mathbf{w})]_i^2, i \in [d_{\text{layer}}]$, 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. Similar figures for Mistral-7B and OPT-6.7B are in Figure 7, and a closer inspection on its relation with weight type and layer depth is in Figure 9 and 10 in Appendix F.2. **We observe that roughly 0.1% parameters in all linear layers contribute about 50% gradient norm square $\|\nabla \mathcal{F}(\mathbf{w})\|_2^2$.**

This idea of sensitive parameters has been studied in the quantization community (Kim et al., 2024; Guo et al., 2023) and FO optimization (Sung et al., 2021). However, *we are the first one to leverage the extremely sparse sensitive parameters in LLM fine-tuning to accelerate ZO fine-tuning with LLMs*. When we have perturbation and updating in the scale of billion parameters, finding which parameters to fine-tune would be important for improving ZO performance. Notice that here we use sensitive masks $\mathbf{m}_k$ for understanding purposes. In Section 3.2, we will discuss how to transform Definition 4 to a parameter-efficient optimization pipeline *via transferable static sparsity*.

2.3 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 C.2.

Theorem 1 (Convergence rate of sensitive sparse ZO-SGD (Definition 4)). *If we pick* $\eta_t = 1/(L(k+2))$*, under Assumptions 1 (bounded gradient error), 2 (Lipschitz smoothness), and 4 (sparse sensitive parameters), we would have*

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)} \|\nabla_{\mathbf{w}} \mathcal{F}(\mathbf{w}_t)\|^2 \leq O\left(\frac{k}{c}\cdot\frac{L}{T}\right)(\mathcal{F}(\mathbf{w}_0) - \mathcal{F}^*) + 3\sigma^2. \tag{5}$$

If we still pick $\eta_t = 1/(L(k+2))$*, with an extra Assumption 3 (P.L. condition), we would have*

$$\mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)}\{\mathcal{F}(\mathbf{w}_T) - \mathcal{F}^*\} \leq \left(1 - O\left(\frac{\mu}{L}\cdot\frac{c}{k}\right)\right)^T (\mathcal{F}(\mathbf{w}_0) - \mathcal{F}^*) + \frac{3\sigma^2 c}{2L(k+2)}. \tag{6}$$

---

[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.

The proof for Inequality 5 is in Appendix C.2 and the proof for Inequality 6 is in Appendix C.3. 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 4 can be easily observed empirically in Figure 7. Liu et al. (2024a) and Ohta et al. (2020) also provide analysis on the convergence of sparse ZO optimization but they do not include our sensitive sparse masks in their studies.

3 SENSZOQ: A SPARSE ON-DEVICE FINE-TUNING RECIPE

In this section, we describe the transferable static sparsity pattern we observed in LLMs and how we utilize it for developing an on-device fine-tuning pipeline of LLMs as SensZOQ.

3.1 TRANSFERABILITY OF STATIC SENSITIVE PARAMETER

Transferable sensitive gradient features: from dynamic to static sparsity. Our Theorem 1 focuses on *dynamic* sparse fine-tuning. However, Panigrahi et al. (2023) notice that in real LLM fine-tuning scenario, the fine-tuning performance could be attributed to a sparse subset of weights ($\sim 0.01\%$). Malladi et al. (2023b) also find certain fine-tuning tasks would demonstrate kernel behaviors, which include "fixed (gradient) features": $\nabla_{\mathbf{w}} f(\mathbf{w}_{\text{after FT}}; (\mathbf{x}, y)) \sim \nabla_{\mathbf{w}} f(\mathbf{w}_{\text{before FT}}; (\mathbf{x}, y))$.

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. This hypothesis can be validated by Figure 3 and Figure 5b. In Figure 3, the fact that "task grad, static" does *not* vanish and still has a large ratio over "task grad, dyn." at the end of training demonstrate that we can select parameters *before fine-tuning*.

Surrogate static sensitive parameter mask. Another observation from Figure 3 is that the sensitive parameters derived from pre-training datasets (C4) would still cover a large fraction of model sensitivity. Specifically, the parameters overlap between top C4 gradient entries and task gradient entries are much (>20×) higher than all weight magnitude baselines. Therefore, we could use it as a *surrogate* sensitive sparse mask when gradients on downstream tasks are unavailable, particularly in scenario of *on-device personalization*. [3]

C4 covers a diverse set of text corpus across different domains and we believe it will produce a *generally good* transferable static mask. We also note that if we have better knowledge on the exact downstream domain or task our method will be applied to, we can extract sparse masks with better specific task performance. So we include an ablation study on other surrogate sensitive parameter masks in Table 6 and we dive deeper into the overlap of top gradient features in Appendix F.4.

3.2 SENSZOQ: AN OPPORTUNITY FOR ON-DEVICE LLM PERSONALIZATION

Transferable static sparse fine-tuning as a parameter-efficient optimization method. 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:

$$\mathbf{w}_{\text{sparse}} = \mathbf{w} \odot \mathbf{m}_k, \;\; \mathbf{w}_{\text{dense}} = \mathbf{w} \odot (\mathbf{1}_d - \mathbf{m}_k) \tag{7}$$

---

[3] Obtaining gradients of LLMs on edge devices is expensive, and we usually cannot transfer data from edge devices to the cloud to compute the gradient on downstream tasks on cloud. In this case we would need some surrogate gradient information to derive sensitive sparse masks on cloud. We will discuss this in Section 3.2.

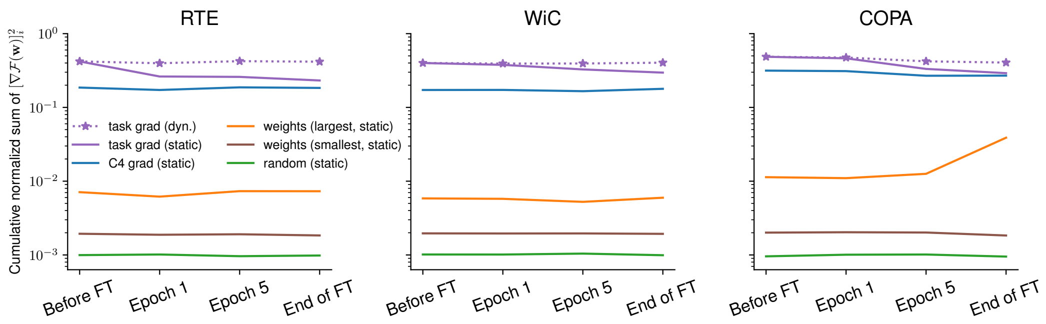


Figure 3: Cumulative normalized gradient square values of Llama2-7B model's linear layers after we apply 99.9% sparsity masks (0.1% nonzeros) of each method. "task grad (dyn.)" refers to the sensitive parameters selected at the given timestep (x-axis) on each downstream dataset, and "task grad (static)" 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 (Raffel et al., 2019), and we keep it unchanged during fine-tuning. "weights (largest, static)" and "weights (smallest, static)" mean we pick the weights with the largest/smallest magnitude in the pre-trained model respectively. We provide a longer discussion in Appendix F.3.

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:

$$\mathbf{w}_{\text{sparse},t+1} = \mathbf{w}_{\text{sparse},t} - \eta_t \hat{g}(\mathbf{w}_{\text{sparse},t}, (\mathbf{x}_t, y_t), \mathbf{z}_{k,t}) \tag{8}$$

Equation 8's formulation allows (1) a wall-clock efficient of ZO optimization method (2) an opportunity to enable an on-device personalization workflow.

Wall-clock time efficiency from extreme static sparsity (more on Appendix H).

- **ZO fine-tuning.** A typical implementation of ZO fine-tuning procedures[4] involve 2 forward calls for SPSA estimator (Definition 1), 3 perturbation calls (2 for SPSA, 1 for resetting the parameters), and 1 optimizer step calls. Under the extreme sparsity regime (only optimize 0.1% parameters), *sensitive ZO optimization on fixed parameters would nearly eliminate the perturbation and optimizer step calls*, which yields 1.2 - 2.5× speedup compared to ZO full fine-tuning.
- **Token generation.** Sensitive ZO would need to optimize far less parameters (0.1%) to reach the same performance as other sparsity methods (will be shown in Figure 5a). This naturally leads to less inference latency and higher throughput during the token generation process.

SensZOQ: integrating sensitive sparse ZO fine-tuning with quantization. As LLMs are often pre-trained with user-agnostic public datasets, personalizing LLMs with individual user's preferences and meet user's specific needs before real-world deployment are vital (Tan et al., 2024a; Mairittha et al., 2020). However, transferring the user-specific data to upstream cloud before fine-tuning LLMs would raise privacy concerns (Xu et al., 2018). On the other hand, personal devices usually have less computational budget and are more memory-constrained than the cloud (Zhu et al., 2023), and performing full fine-tuning would easily exceed the device memory budget.

In response, we propose an on-device personalization workflow **SensZOQ** illustrated in Figure 4. 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).

We highlight that SensZOQ's memory consumption is *nearly minimal*: we can fine-tune a Llama2-7B model under 8 GiB GPU memory *without any offloading* as illustrated in Figure 1. This would satisfy the memory constraint by a wide range of edge or mobile devices as illustrated in Table 11.

---

[4] We take MeZO's implementation (Malladi et al., 2023a) as a reference.

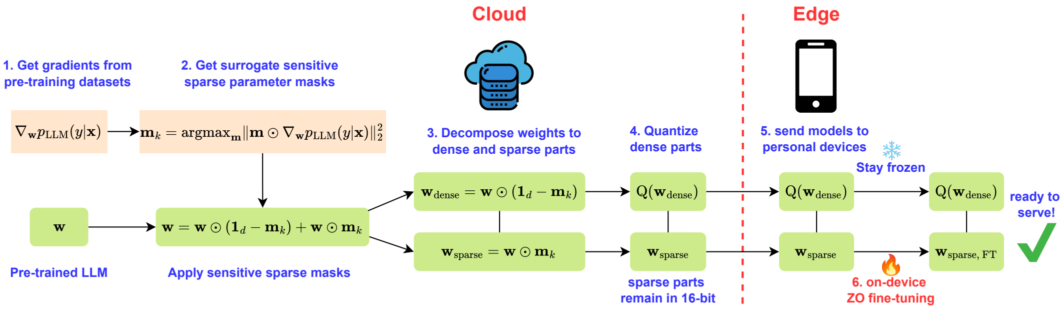

Cloud
Edge
1. Get gradients from pre-training datasets
2. Get surrogate sensitive sparse parameter masks
$\nabla_{\mathbf{w}} p_{\text{LLM}}(y|\mathbf{x})$
$\mathbf{m}_k = \text{argmax}_{\mathbf{m}} \|\mathbf{m} \odot \nabla_{\mathbf{w}} p_{\text{LLM}}(y|\mathbf{x})\|_2^2$
3. Decompose weights to dense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

Figure 4: **SensZOQ**: an on-device LLM personalization workflow via integrating **Sens**itive **ZO** optimization with **Q**uantization. The high-level idea is that we first decompose $\mathbf{w}$ to $\mathbf{w}_{\text{sparse}}$ and $\mathbf{w}_{\text{dense}}$, and then quantize $\mathbf{w}_{\text{dense}}$ (on the cloud). On the edge devices, we will fine-tune $\mathbf{w}_{\text{sparse}}$ only.

In addition, our method does *not* put strict constraints on specific choices of quantization algorithms since any algorithm that aims to minimize the least-square quantization error term $Q(\mathbf{w}) = \text{argmin}_{Q(\mathbf{w})} \mathbb{E}_{\mathbf{x}} \|(\mathbf{w} - Q(\mathbf{w}))\mathbf{x}\|_2^2$ or its variant would suffice (Chee et al., 2024; Nagel et al., 2020; Frantar et al., 2022; Lin et al., 2023; Kim et al., 2024).

Efficient implementation of sensitive sparse linear layers. In Appendix G, we discuss how to efficiently implement our sensitive sparse ZO fine-tuning in forward passes of linear layers with training and inference workflow: use Equation 18 when we have access to efficient uniform integer matmul to compute $Q(\mathbf{w}_{\text{dense}})\mathbf{x}$ and in other cases, we would usually use Equation 19 for token generation and Equation 21 for ZO training.

4 EXPERIMENTS

In this section, we aim to validate the effectiveness of our SensZOQ, shown in Figure 4, as a memory-efficient LLM fine-tuning solution. This naturally leads to comparison with other ZO methods, which we evaluate in Section 4.1. Additionally, we assess the effectiveness of our sensitive parameter mask derived from pre-training texts (C4) against other heuristic sparsity methods in Section 4.2. Specifically, we aim to address the following research questions:

- **RQ1:** What is the performance of our SensZOQ compared with other ZO methods?
- **RQ2:** Is optimizing C4-gradient-derived sensitive parameters more effective than optimizing other subset of parameters during ZO fine-tuning?

We focus on 7B-level LLM models including Llama2-7B (Touvron et al., 2023), Mistral-7B (Jiang et al., 2023), and OPT-6.7B (Zhang et al., 2022) as they would fit with common on-device memory constraints (8 GiB) listed on Table 11 after applying quantization. We use SST-2 (Socher et al., 2013), RTE (Wang et al., 2018), CB (De Marneffe et al., 2019), BoolQ (Clark et al., 2019), WSC (Levesque et al., 2012), WiC (Pilehvar & Camacho-Collados, 2019), COPA (Roemmele et al., 2011), WinoGrande (WinoG) (Sakaguchi et al., 2020), and WikiText-2 (Wiki2) (Merity et al., 2017) datasets. We follow standard ZO fine-tuning settings and use the same codebase as in Malladi et al. (2023a). More details of our experiments (hyperparameters, task-specific prompts, etc.) are in Appendix I.

We include additional results for Llama2-13B and OPT-13B, and harder tasks such as commonsense reasoning, math reasoning, and MMLU in Appendix D. **To the best of our knowledge, there are no ZO-LLM research yet evaluated on harder commonsense reasoning or math tasks. We take a pioneering step in this direction and establish the ZO baselines.** In addition, we investigate the empirical training convergence w.r.t. optimization steps for OPT-13B in Appendix E.

4.1 ON-DEVICE PERSONALIZATION

We evaluate the performance of our SensZOQ method in Table 1. We follow the exact recipe as described Figure 4, where we only optimize 0.1% sensitive parameters derived from a small batch of C4 texts on top of a 4-bit quantized model. SensZOQ's results are shown as 1^{st} row for each subtable.

Table 1: Fine-tuning performance of different methods. In the first column, "Q" means the full model is quantized with 4-bit quantization method (SqueezeLLM (Kim et al., 2024)), and "ZO-SGD" means the model is fine-tuned with ZO-SGD optimizer. For each cell, we report the mean and standard deviation of test set accuracy (↑) of 3 random trials in the format of mean_{std}, except that we report test set perplexity (↓) for Wiki2. We finally report the average performance rank (Wiki2 included) and average accuracy (Wiki2 excluded) in the last 2 columns. Notice that we add the results for FO-SGD & FO-Adam for reference and we do not use it for performance rank computation.

(a) **Llama2-7B**

	Methods	SST-2	RTE	CB	BoolQ	WSC	WiC	COPA	WinoG	Wiki2	**Rank↓**	**Acc↑**
Q, ZO-SGD	**SensZOQ**	$94.7_{0.4}$	$\mathbf{74.7}_{1.2}$	$66.7_{2.2}$	$\mathbf{83.0}_{0.5}$	$57.4_{3.9}$	$\mathbf{65.2}_{0.9}$	$85.0_{2.2}$	$\mathbf{65.7}_{0.7}$	$\mathbf{5.39}_{.005}$	**1.9**	**74.1**
	LoRA	$93.8_{0.6}$	$64.7_{1.1}$	$64.9_{4.7}$	$79.7_{1.1}$	$\mathbf{61.5}_{2.1}$	$59.8_{0.1}$	$\mathbf{85.7}_{0.5}$	$63.8_{0.6}$	$5.42_{.004}$	3.4	71.7
	Prefix	$80.5_{4.3}$	$65.5_{1.2}$	$63.1_{3.0}$	$80.3_{0.2}$	$54.5_{11.4}$	$58.3_{1.3}$	$82.0_{0.8}$	$62.6_{0.5}$	$5.74_{.016}$	4.6	68.4
ZO-SGD	Full FT	$94.6_{0.5}$	$73.3_{5.1}$	$66.7_{0.8}$	$81.9_{0.8}$	$58.0_{4.3}$	$61.9_{0.2}$	$82.7_{1.7}$	$63.1_{0.4}$	$5.42_{.002}$	2.8	72.2
	Zero-shot	$89.0_{0.0}$	$57.8_{0.0}$	$32.1_{0.0}$	$69.9_{0.2}$	$50.2_{0.0}$	$36.5_{0.0}$	$79.0_{0.0}$	$64.8_{0.6}$	$5.48_{.000}$	5.4	59.9
	ICL	$\mathbf{94.8}_{0.2}$	$71.5_{4.3}$	$\mathbf{72.6}_{15.2}$	$77.5_{4.6}$	$53.2_{1.1}$	$61.1_{4.3}$	$87.0_{2.2}$	$67.5_{1.3}$	NA	2.5	73.2
FO-SGD	Full FT	$95.4_{0.3}$	$84.1_{0.9}$	$73.2_{0.0}$	$85.1_{1.2}$	$62.8_{0.5}$	$72.0_{1.8}$	$85.3_{1.2}$	$71.1_{1.7}$			78.6
FO-Adam	Full FT	$96.0_{0.4}$	$85.1_{0.3}$	$86.9_{9.9}$	$85.5_{0.7}$	$57.7_{3.6}$	$71.8_{0.4}$	$87.7_{0.5}$	$79.2_{1.0}$	$4.83_{.000}$		81.2

(b) **Mistral-7B**

Q, ZO-SGD	**SensZOQ**	$94.0_{0.3}$	$\mathbf{78.0}_{0.6}$	$\mathbf{70.2}_{2.2}$	$75.1_{2.4}$	$59.6_{4.9}$	$\mathbf{63.6}_{0.7}$	$\mathbf{88.3}_{1.2}$	$\mathbf{74.1}_{0.5}$	$5.24_{.002}$	**1.7**	**75.4**
	LoRA	$94.0_{0.4}$	$65.3_{1.3}$	$64.9_{4.5}$	$70.3_{3.7}$	$\mathbf{60.9}_{3.7}$	$61.1_{0.4}$	$\mathbf{88.3}_{0.5}$	$71.2_{1.2}$	$5.27_{.004}$	2.9	72.0
	Prefix	$86.9_{2.1}$	$57.3_{1.4}$	$63.7_{5.9}$	$62.2_{0.9}$	$60.3_{4.6}$	$49.0_{0.3}$	$81.3_{1.7}$	$64.2_{1.3}$	$5.44_{.003}$	4.0	65.6
ZO-SGD	Full FT	$\mathbf{94.6}_{0.1}$	$74.6_{2.1}$	$68.8_{6.2}$	$\mathbf{76.6}_{0.2}$	$54.8_{6.2}$	$62.6_{0.5}$	$\mathbf{88.3}_{0.5}$	$72.2_{0.5}$	$\mathbf{5.23}_{.004}$	2.1	74.1
	Zero-shot	$54.8_{0.0}$	$50.5_{0.0}$	$37.5_{0.0}$	$43.4_{1.8}$	$50.8_{0.0}$	$39.4_{0.0}$	$78.0_{0.0}$	$66.2_{0.1}$	$5.25_{.000}$	5.0	52.6
	ICL	$60.7_{16.7}$	$55.2_{4.7}$	$33.3_{13.1}$	$46.8_{6.5}$	$50.4_{0.6}$	$63.8_{0.9}$	$88.7_{0.5}$	$74.0_{0.8}$	NA	3.9	59.1
FO-SGD	Full FT	$94.9_{0.6}$	$87.6_{1.2}$	$85.7_{3.9}$	$86.1_{0.7}$	$62.5_{0.0}$	$70.8_{0.6}$	$88.3_{1.7}$	$82.1_{1.1}$			82.3
FO-Adam	Full FT	$95.1_{0.2}$	$86.4_{0.7}$	$88.1_{3.4}$	$83.1_{1.5}$	$64.7_{7.3}$	$72.7_{2.9}$	$82.7_{1.7}$	$85.9_{0.3}$	$4.81_{.013}$		82.3

(c) **OPT-6.7B**

Q, ZO-SGD	**SensZOQ**	$\mathbf{94.9}_{0.5}$	$\mathbf{72.8}_{3.6}$	$\mathbf{83.3}_{5.1}$	$\mathbf{73.9}_{0.7}$	$59.3_{5.3}$	$\mathbf{62.0}_{2.0}$	$\mathbf{84.0}_{1.4}$	$\mathbf{65.0}_{0.8}$	$\mathbf{9.82}_{.009}$	**1.1**	**74.4**
	LoRA	$94.2_{0.2}$	$69.6_{1.6}$	$69.0_{1.7}$	$69.6_{2.0}$	$57.1_{9.1}$	$57.2_{0.8}$	$83.0_{2.2}$	$63.1_{0.4}$	$9.90_{.000}$	3.4	70.4
	Prefix	$93.3_{0.4}$	$71.2_{1.0}$	$72.0_{1.7}$	$68.9_{2.8}$	$\mathbf{62.5}_{2.4}$	$59.4_{0.5}$	$80.0_{2.4}$	$63.7_{1.1}$	$9.92_{.033}$	3.3	71.4
ZO-SGD	Full FT	$94.4_{0.3}$	$72.7_{1.2}$	$79.8_{3.0}$	$72.1_{1.2}$	$57.4_{4.6}$	$60.2_{0.9}$	$82.3_{2.6}$	$64.6_{0.3}$	$9.88_{.009}$	2.2	72.9
	Zero-shot	$61.0_{0.0}$	$60.7_{0.0}$	$46.4_{0.0}$	$55.7_{1.0}$	$55.5_{0.0}$	$36.5_{0.0}$	$77.0_{0.0}$	$61.1_{0.3}$	$10.88_{.000}$	5.8	56.7
	ICL	$74.0_{14.6}$	$65.8_{11.2}$	$54.8_{5.9}$	$67.9_{2.1}$	$53.2_{1.7}$	$41.0_{4.5}$	$80.7_{2.9}$	$61.5_{0.8}$	NA	5.0	62.4
FO-SGD	Full FT	$95.2_{0.3}$	$81.8_{0.9}$	$92.3_{3.0}$	$79.2_{1.3}$	$59.0_{7.7}$	$66.5_{2.3}$	$85.7_{0.9}$	$68.8_{0.6}$			78.6
FO-Adam	Full FT	$95.7_{0.2}$	$81.1_{2.6}$	$83.9_{3.9}$	$81.1_{0.7}$	$56.1_{7.9}$	$66.5_{0.5}$	$81.3_{1.2}$	$66.4_{0.8}$	$8.51_{.000}$		76.5

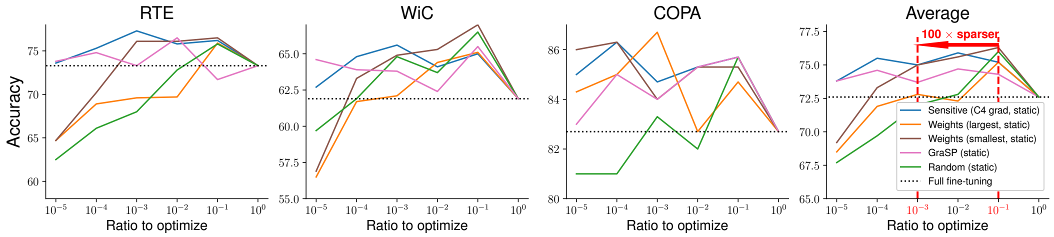
Comparison with ICL & ZO Full FT. The results of in-context learning (ICL) and ZO full fine-tuning (ZO Full FT) on 16-bit models are shown as the 4th and 5th row for each substable in Table 1. 7B models are usually not large enough such that ICL would outperform with FT (Liu et al., 2022; Mosbach et al., 2023). In addition, ICL induces additional KV-cache memory burden if the number of demonstration examples are large or demonstration texts are long. Our method SensZOQ *does not* induce significant inference latency and memory burden as it only needs to use 0.1% parameters. SensZOQ also outperform ICL and ZO Full FT. This is impressive give that quantization would degrade performance of the base model for SensZOQ, but SensZOQ still manages to match the fine-tuning performance of 16-bit model.

Comparison with ZO PEFT methods. The primary purpose of quantization is to represent parameters in less bits (therefore reducing model sizes) and improve system-level metrics such as weight loading time and inference latency (Dettmers et al., 2022; Chee et al., 2024). In order to retain such benefits during fine-tuning stage, our fine-tuning methods should be parameter-efficient. A natural baseline becomes fine-tuning PEFT methods such as LoRA (Hu et al., 2021) and Prefix Tuning (Li & Liang, 2021) on top of the same quantized LLM weights as SensZOQ. These results are shown as the 2nd and 3rd row for each substable in Table 1. SensZOQ still outperforms both LoRA and Prefix Tuning when applied to the same 4-bit quantized base model.

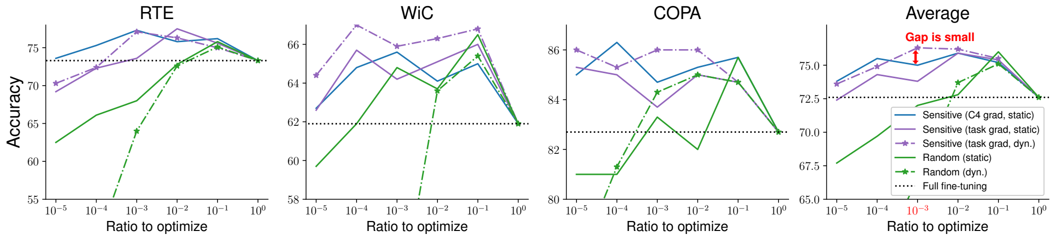
Comparison with Adam Full FT of smaller size model. The memory-efficiency of ZO is achieved via cheap queries in the loss landscape with random perturbation. The absence of accurate descent direction naturally leads an inferior performance than first-order (FO) full FT on the same model size, as also observed by Malladi et al. (2023a). However, we argue that such performance degradation is still acceptable in terms that *ZO FT on larger models outperform FO FT on smaller models*. In Table 2, we pick 2 popular LLM sizes (1B and 7B scale) and we find that applying SensZOQ on 6.7B OPT model generally outperforms FO-Adam full FT on 1.3B OPT model, and the later already surpasses the 8 GiB memory budget (e.g., 11.6 GiB on RTE compared to SensZOQ's 5.2 GiB).

Table 2: Fine-tuning performance of SensZOQ versus Adam FT of smaller model in the OPT family. We follow the same experiment procedure as in Table 1.

	# Params	**Methods**	SST-2	RTE	CB	BoolQ	WSC	WiC	COPA	WinoG	Wiki2	**Rank↓**	**Acc↑**
4-bit ZO-SGD	6.7B	**SensZOQ**	**94.9**$_{0.5}$	72.8$_{3.6}$	**83.3**$_{5.1}$	**73.9**$_{0.7}$	59.3$_{5.3}$	62.0$_{2.0}$	**84.0**$_{1.4}$	**65.0**$_{0.8}$	**9.82**$_{.009}$	**1.3**	**74.4**
16-bit ZO-SGD	6.7B	Full FT	94.4$_{0.3}$	72.7$_{1.2}$	79.8$_{3.0}$	72.1$_{1.2}$	57.4$_{4.6}$	60.2$_{0.9}$	82.3$_{2.6}$	64.6$_{0.3}$	9.88$_{.009}$	2.4	72.9
16-bit FO-Adam	1.3B	Full FT	93.6$_{0.5}$	**73.9**$_{2.4}$	75.0$_{3.9}$	73.8$_{1.0}$	**61.9**$_{1.2}$	**62.4**$_{1.5}$	76.7$_{1.2}$	60.8$_{0.6}$	10.75$_{.000}$	2.2	72.3



(a) Static transferability performance of different sparsity masks. "static" means that we will determine the trainable parameters (sparsity mask) before fine-tuning and other parameters are kept unchanged.



(b) The performance gap between sensitive parameters derived from causal LM loss in C4 datasets and gradients from each fine-tuning task. "Static" means the parameters to optimize are determined before fine-tuning and other parameters are kept unchanged during fine-tuning (static sparsity). "Dyn." means the parameters to optimize will be updated every 100 training steps (dynamic sparsity).

Figure 5: Performance of different sparsity methods in Llama2-7B ZO fine-tuning.

4.2 EFFECTIVENESS OF SPARSE ZO FINE-TUNING ON SENSITIVE PARAMETERS

Comparison with other static sparsity masks. We first investigate the performance of optimizing our sensitive parameters versus other subsets of parameters in static sparsity regime with the 16-bit model. We first consider standard weight-magnitude baselines as weights with largest magnitude, weights with smallest magnitude (SparseMeZO's sparsity patterns (Liu et al., 2024a)), and a random subset of weights baseline. There are some other weight importance metrics in the pruning community such as GraSP (Wang et al., 2020), and we also evaluate their performance in the static transfer setting. *We note that these pruning metrics were originally proposed for deciding which parameters to retain instead of being removed during pruning, and it has no direct implications for ZO FT*. Given a threshold vector τ, the formal definitions of all methods are listed as below:

- sensitive parameters with C4 gradients: $\mathbf{m} = |\nabla f(\mathbf{w}_{\text{before FT}}; (\mathbf{x}_{\text{C4}}, y_{\text{C4}}))| \geq \tau$
- random subsets: $\mathbf{m} = \text{random\_dim\_d\_vector\_with\_k\_nnz}(d, k)$
- weights with largest magnitude: $\mathbf{m} = |\mathbf{w}_{\text{before FT}}| \geq \tau$
- weights with smallest magnitude: $\mathbf{m} = |\mathbf{w}_{\text{before FT}}| \leq \tau$
- smallest GraSP scores: $\mathbf{m} = -\mathbf{w}_{\text{before FT}} \odot \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}} H(\mathbf{w}_{\text{before FT}}) \nabla_{\mathbf{w}} f(\mathbf{w}_{\text{before FT}}) \leq \tau$

As illustrated in Figure 5a, we can find that ZO fine-tuning would benefit from sparse optimization, as all methods would achieve higher than ZO full fine-tuning when optimizing 10% parameters. However, only sensitive parameters would maintain its performance as we move to the extreme sparsity region (<1%). *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 10% to 0.1% is minimal. We also find that optimizing weights with smallest magnitude is more effectively than optimizing weights with largest magnitude, which aligns with Liu et al. (2024a)'s findings. However, sensitive parameters is still more effective for optimizing weights with smallest magnitude. This suggests that sensitive parameters are more effective to serve as the sparse parameters in our SensZOQ instead of SparseMeZO's weights with smallest magnitude.

Transferability of SensZOQ's sparsity masks. SensZOQ uses C4 gradients to produce a transferable static mask used in downstream tasks. In Figure 5b, we compare the performance of optimizing sensitive parameters with gradients on C4 dataset with its theoretical upper bound: static sensitive parameters derived from gradients on each fine-tuning task as the solid line and its dynamic version as the dash-dotted line. We also include the static and dynamic random subset parameters as a baseline. We can find that the gap of sensitive parameters between deriving from gradients on C4 dataset and gradients on each fine-tuning task at ratio 1e-3 is *small*. Together with Figure 11 that we evaluate the top gradient entries similarity between C4 and downstream tasks, we believe SensZOQ's sensitive masks from C4 gradients would yield satisfactory performance in general.

5 RELATED WORKS

Zeroth-order fine-tuning of LLMs. Since MeZO (Malladi et al., 2023a) first demonstrates the effectiveness of ZO for LLM fine-tuning, ZO has attracted great research interests from the LLM community in different aspects. For example, ZO is effective for edge device fine-tuning in communication-constrained and federated settings (Ling et al., 2024; Zhang et al., 2024a; Tang et al., 2024; Liu et al., 2024b). Notably, Zelikman et al. (2023) illustrate the possibility of exchanging single-byte projected gradients in distributed zeroth-order workloads, yielding both communication and privacy benefits. Numerous research has also focused on enhancing the optimizer aspect of zeroth-order optimization (Jiang et al., 2024; Pang & Zhou, 2024; Gautam et al., 2024). Other researchers are also interested in improving ZO's efficiency or convergence rate from cleverer optimizer designs. Li et al. (2024b) explore the middle ground between small-batched FO-SGD and large-batched ZO-SGD to balance the convergence speed and memory footprints. Liu et al. (2024a) and Zhang et al. (2024b) suggest that sparsity would potentially accelerate ZO optimization convergence. Zhao et al. (2024) precondition ZO perturbation with knowledge from parameter-wise loss curvature heterogeneity to gain convergence speedup. To tbe best of our knowledge, Liu et al. (2024a) is the only ZO fine-tuning with sparsity (smallest weight magnitude mask) work at this moment, and we have ablated on its static extreme sparsity performance in Figure 5a.

6 CONCLUSION

In this work, we identify that only a small portion of LLM parameters needs to be updated during ZO fine-tuning, and these static and sparse subset parameters can be derived during the pre-training phase and transferred across various downstream tasks without requiring any modifications, preserving efficient ZO performance. We propose SensZOQ, a workflow that integrates sparse ZO optimization with 4-bit quantization to further enhance the memory efficiency of on-device fine-tuning. SensZOQ leverages static sparse fine-tuning to enable the personalization of 7B LLMs on-device, reducing memory consumption to less than 8 GiB of CUDA memory. Despite this efficiency, SensZOQ achieves better performance than both in-context learning (ICL) and full ZO fine-tuning. Therefore, SensZOQ creates a new venue to facilitate on-device fine-tuning.

REFERENCES

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. Quip: 2-bit quantization of large language models with guarantees. *Advances in Neural Information Processing Systems*, 36, 2024.

Aochuan Chen, Yimeng Zhang, Jinghan Jia, James Diffenderfer, Jiancheng Liu, Konstantinos Parasyris, Yihua Zhang, Zheng Zhang, Bhavya Kailkhura, and Sijia Liu. Deepzero: Scaling up zeroth-order optimization for deep model training. In *International Conference on Learning Representations*, 2024. doi: 10.48550/arXiv.2310.02025.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, 2019.

Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. A discourse-aware attention model for abstractive summarization of long documents. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 2018. doi: 10.18653/v1/n18-2097. URL http://dx.doi.org/10.18653/v1/n18-2097.

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

Marie-Catherine De Marneffe, Mandy Simons, and Judith Tonhauser. The commitmentbank: Investigating projection in naturally occurring discourse. In *Proceedings of Sinn und Bedeutung*, pp. 107–124, 2019.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35: 30318–30332, 2022.

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

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

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Tanmay Gautam, Youngsuk Park, Hao Zhou, Parameswaran Raman, and Wooseok Ha. Variance-reduced zeroth-order methods for fine-tuning language models, 2024.

In Gim and JeongGil Ko. 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.

Han Guo, Philip Greengard, Eric Xing, and Yoon Kim. Lq-lora: Low-rank plus quantized matrix decomposition for efficient language model finetuning. In *The Twelfth International Conference on Learning Representations*, 2023.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Lee. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5254–5276, 2023.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Shuoran Jiang, Qingcai Chen, Youcheng Pan, Yang Xiang, Yukang Lin, Xiangping Wu, Chuanyi Liu, and Xiaobao Song. Zo-adamu optimizer: Adapting perturbation by the momentum and uncertainty in zeroth-order optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):18363–18371, Mar. 2024. doi: 10.1609/aaai.v38i16.29796. URL `https://ojs.aaai.org/index.php/AAAI/article/view/29796`.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Sehoon Kim, Coleman Richard Charles Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. In *Forty-first International Conference on Machine Learning*, 2024.

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

Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Thirteenth international conference on the principles of knowledge representation and reasoning*, 2012.

Luchang Li, Sheng Qian, Jie Lu, Lunxi Yuan, Rui Wang, and Qin Xie. Transformer-lite: High-efficiency deployment of large language models on mobile phone gpus. *arXiv preprint arXiv:2403.20041*, 2024a.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

Zeman Li, Xinwei Zhang, and Meisam Razaviyayn. Addax: Memory-efficient fine-tuning of language models with a combination of forward-backward and forward-only passes. In *5th Workshop on practical ML for limited/low resource settings*, 2024b.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *ACL*, 2017.

Zhenqing Ling, Daoyuan Chen, Liuyi Yao, Yaliang Li, and Ying Shen. On the convergence of zeroth-order federated tuning for large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, pp. 1827–1838, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704901. doi: 10.1145/3637528.3671865. URL `https://doi.org/10.1145/3637528.3671865`.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965, 2022.

Sijia Liu, Pin-Yu Chen, Bhavya Kailkhura, Gaoyuan Zhang, Alfred O Hero III, and Pramod K Varshney. A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications. *IEEE Signal Processing Magazine*, 37(5):43–54, 2020.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Yong Liu, Zirui Zhu, Chaoyu Gong, Minhao Cheng, Cho-Jui Hsieh, and Yang You. Sparse mezo: Less parameters for better performance in zeroth-order llm fine-tuning. *arXiv preprint arXiv:2402.15751*, 2024a.

Z Liu, J Lou, W Bao, Y Hu, B Li, Z Qin, and K Ren. Differentially private zeroth-order methods for scalable large language model finetuning. *arXiv preprint arXiv:2402.07818*, 2024b. URL https://arxiv.org/abs/2402.07818.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pp. 22137–22176. PMLR, 2023.

Zirui Liu, Guanchu Wang, Shaochen Henry Zhong, Zhaozhuo Xu, Daochen Zha, Ruixiang Ryan Tang, Zhimeng Stephen Jiang, Kaixiong Zhou, Vipin Chaudhary, Shuai Xu, et al. Winner-take-all column row sampling for memory efficient adaptation of language model. *Advances in Neural Information Processing Systems*, 36, 2024c.

Nattaya Mairittha, Tittaya Mairittha, and Sozo Inoue. 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.

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems*, 36:53038–53075, 2023a.

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

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2016.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.

Marius Mosbach, Tiago Pimentel, Shauli Ravfogel, Dietrich Klakow, and Yanai Elazar. Few-shot fine-tuning vs. in-context learning: A fair comparison and evaluation. *arXiv preprint arXiv:2305.16938*, 2023.

Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*, pp. 7197–7206. PMLR, 2020.

Mayumi Ohta, Nathaniel Berger, Artem Sokolov, and Stefan Riezler. 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.

Yijiang Pang and Jiayu Zhou. Stochastic two points method for deep model zeroth-order optimization. *arXiv preprint arXiv:2402.01621*, 2024. URL https://arxiv.org/abs/2402.01621.

Abhishek Panigrahi, Nikunj Saunshi, Haoyu Zhao, and Sanjeev Arora. Task-specific skill localization in fine-tuned language models. In *International Conference on Machine Learning*, pp. 27011–27033. PMLR, 2023.

Gunho Park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, Dongsoo Lee, et al. Lut-gemm: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models. In *The Twelfth International Conference on Learning Representations*, 2024.

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. Openwebmath: An open dataset of high-quality mathematical web text. In *The Twelfth International Conference on Learning Representations*, 2024.

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

Mohammad Taher Pilehvar and Jose Camacho-Collados. Wic: the word-in-context dataset for evaluating context-sensitive meaning representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 1267–1273, 2019.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv e-prints*, 2019.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*, 2011.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8732–8740, 2020.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 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.

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, Valentin Hofmann, Ananya Jha, Sachin Kumar, Li Lucy, Xinxi Lyu, Nathan Lambert, Ian Magnusson, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew Peters, Abhilasha Ravichander, Kyle Richardson, Zejiang Shen, Emma Strubell, Nishant Subramani, Oyvind Tafjord, Evan Walsh, Luke Zettlemoyer, Noah Smith, Hannaneh Hajishirzi, Iz Beltagy, Dirk Groeneveld, Jesse Dodge, and Kyle Lo. Dolma: an open corpus of three trillion tokens for language model pretraining research. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15725–15788, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.840. URL https://aclanthology.org/2024.acl-long.840.

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

Yi-Lin Sung, Varun Nair, and Colin A Raffel. Training neural networks with fixed sparse masks. *Advances in Neural Information Processing Systems*, 34:24193–24205, 2021.

Zhaoxuan Tan, Qingkai Zeng, Yijun Tian, Zheyuan Liu, Bing Yin, and Meng Jiang. Democratizing large language models via personalized parameter-efficient fine-tuning. *arXiv preprint arXiv:2402.04401*, 2024a.

Zhen Tan, Tianlong Chen, Zhenyu Zhang, and Huan Liu. Sparsity-guided holistic explanation for llms with interpretable inference-time intervention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 21619–21627, 2024b.

Xinyu Tang, Ashwinee Panda, Milad Nasr, Saeed Mahloujifar, and Prateek Mittal. Private fine-tuning of large language models with zeroth-order optimization. *arXiv preprint arXiv:2401.04343*, 2024. URL `https://arxiv.org/abs/2401.04343`.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 353–355, 2018.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020.

Haocheng Xi, Changhao Li, Jianfei Chen, and Jun Zhu. Training transformers with 4-bit integers. *Advances in Neural Information Processing Systems*, 36:49146–49168, 2023.

Haojun Xia, Zhen Zheng, Yuchao Li, Donglin Zhuang, Zhongzhu Zhou, Xiafei Qiu, Yong Li, Wei Lin, and Shuaiwen Leon Song. 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.

Mengwei Xu, Feng Qian, Qiaozhu Mei, Kang Huang, and Xuanzhe Liu. 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.

Xinyu Yang, Jixuan Leng, Geyang Guo, Jiawei Zhao, Ryumei Nakada, Linjun Zhang, Huaxiu Yao, and Beidi Chen. s^2ft: Efficient, scalable and generalizable llm fine-tuning by structured sparsity. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Eric Zelikman, Qian Huang, Percy Liang, Nick Haber, and Noah D. Goodman. Just one byte (per gradient): A note on low-bandwidth decentralized language model finetuning using shared randomness. *arXiv preprint arXiv:2306.10015*, 2023. URL `https://arxiv.org/abs/2306.10015`.

Liang Zhang, Bingcong Li, Kiran Koshy Thekumparampil, Sewoong Oh, and Niao He. Dpzero: Private fine-tuning of language models without backpropagation. *arXiv preprint arXiv:2310.09639*, 2024a. URL `https://arxiv.org/abs/2310.09639`.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Yihua Zhang, Pingzhi Li, Junyuan Hong, Jiaxiang Li, Yimeng Zhang, Wenqing Zheng, Pin-Yu Chen, Jason D Lee, Wotao Yin, Mingyi Hong, et al. Revisiting zeroth-order optimization for memory-efficient llm fine-tuning: A benchmark. *arXiv preprint arXiv:2402.11592*, 2024b.

Yanjun Zhao, Sizhe Dang, Haishan Ye, Guang Dai, Yi Qian, and Ivor W Tsang. Second-order fine-tuning without pain for llms: A hessian informed zeroth-order optimizer. *arXiv preprint arXiv:2402.15173*, 2024.

Shaochen Zhong, Guanqun Zhang, Ningjia Huang, and Shuai Xu. Revisit kernel pruning with lottery regulated grouped convolutions. In *International Conference on Learning Representations*, 2021.

Shaochen Henry Zhong, Zaichuan You, Jiamu Zhang, Sebastian Zhao, Zachary LeClaire, Zirui Liu, Daochen Zha, Vipin Chaudhary, Shuai Xu, and Xia Hu. One less reason for filter pruning: Gaining free adversarial robustness with structured grouped kernel pruning. *Advances in Neural Information Processing Systems*, 36, 2024.

Ligeng Zhu, Lanxiang Hu, Ji Lin, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. Pockengine: Sparse and efficient fine-tuning in a pocket. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 1381–1394, 2023.

APPENDIX

In Section A, we discuss some related works of sparsity techniques in LLM. In Section B we describe all notations used in this paper. In Section C, we include the assumptions and exact proof on the convergence rate (Theorem 1). In Section D, we include experiment results on OPT-13B and Llama2-13B. We also take a pioneering move to investigate the effectiveness of SensZOQ and ZO methods in commonsense reasoning, Math, and MMLU tasks. In Section E, we investigate the empirical training convergence w.r.t. optimization steps for OPT-13B. In Section F, we inspect the appearance of sensitive parameters across models and tasks. We also investigate the effects of different data sources that would produce different sensitive masks. In Section G, we 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. In Section H, we investigate the wall-clock time efficiency of our static sparse ZO fine-tuning formulation. In Section I, we describe miscellaneous details (hyperparameters, task templates, hardware config, etc.) in our experiments.

A RELATED WORKS OF SPARSITY IN LLM

Sparsity-driven techniques are widely adopted in improving ML model's efficiency (Tan et al., 2024b; 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) show 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) demonstrate that transformer-based models, such as OPT (Zhang et al., 2022), exhibit great sparsity ($\geq 95\%$) in activations. Moreover, Panigrahi et al. (2023) discover 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.

B NOTATIONS

We present the notations used in this work as follows.

Table 3: Notations used in this paper

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 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 $(\mathbf{x}, y)$
$\mathcal{F}(\mathbf{w})$	full-batched training loss of $\mathbf{w}$
$H(\mathbf{w}; (\mathbf{x}, y))$	Hessian matrix of $\mathbf{w}$ evaluated at $(\mathbf{x}, y)$
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 1)
η_t	learning rate for ZO-SGD optimizer (Definition 2) at timestep t
$\mathbf{m}_k \in \{0, 1\}^d$	a sensitive sparse mask with k nonzero entries (Definition 3)
$\mathbf{m}_{k,t} \in \{0, 1\}^d$	a sensitive sparse mask with k nonzero entries 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 $\mathbf{m}_k$
$\mathbf{1}_d$	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
L	Lipschitz constant in Assumption 2
μ	PL condition number in Assumption 3
σ^2	stochastic gradient error term in Assumption 1
W_Q	the query weight matrix Q in attention layers
W_K	the key weight matrix K in attention layers
W_V	the value weight matrix V in attention layers
W_O	linear weight matrix for the output embedding matrix O in attention layers
W_{Gate}	the gated unit layer in SwiGLU for Llama architecture
W_{Up}	the up projection weight layer in SwiGLU for Llama architecture
W_{Down}	the down projection weight layer in SwiGLU for Llama architecture

C THEORETICAL CONVERGENCE RATE

C.1 ASSUMPTIONS

We start with listing standard assumptions in nonconvex optimization literature:

Assumption 1 (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*

$$\|\nabla_{\mathbf{w}} f(\mathbf{w}; (\mathbf{x}, y)) - \nabla_{\mathbf{w}} \mathcal{F}(\mathbf{w})\|^2 \leq \sigma^2. \tag{9}$$

Assumption 2 (Lipschitz smoothness). *We assume that* $f(\mathbf{w}, \mathbf{x})$ *is* L*-Lipschitz smooth* ($L > 0$)*: for any* $\mathbf{w}, \mathbf{w}' \in \mathbb{R}^d$*,*

$$\|\nabla_{\mathbf{w}} f(\mathbf{w}; (\mathbf{x}, y)) - \nabla_{\mathbf{w}} f(\mathbf{w}'; (\mathbf{x}, y))\| \leq L\|\mathbf{w} - \mathbf{w}'\|. \tag{10}$$

Assumption 3 (PL inequality). *We assume that* $\mathcal{F}(\mathbf{w})$ *fulfills the Polyak-Lojasiewicz (PL) condition: there exists some* $\mu > 0$*, for any* $\mathbf{w} \in \mathbb{R}^d$

$$\frac{1}{2}\|\nabla_{\mathbf{w}} \mathcal{F}(\mathbf{w})\|^2 \geq \mu(\mathcal{F}(\mathbf{w}) - \mathcal{F}^*), \quad \mathcal{F}^* \textit{ is the minimum value } \mathcal{F}^* = \inf_{\mathbf{w}} \mathcal{F}(\mathbf{w}). \tag{11}$$

Inspired by Figure 7, we would assume the sensitive parameters of $\mathbf{w}$ are sparse.

Assumption 4 (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 $c \gg k/d$*.* [5]

C.2 PROOF FOR EQUATION 5, THEOREM 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 1 (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 1. 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 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:

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

□

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

---

[5] From Figure 7, 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 5, Theorem 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,

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

□

C.3 PROOF FOR EQUATION 6, THEOREM 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 6, Theorem 1. Denote κ as the condition number $\kappa = \frac{\mu}{L}$.

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

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

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

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

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

□

D MORE EXPERIMENT RESULTS

13B model results. In Table 4, we compare SensZOQ vs. ZO full FT & ICL in 2 popular 13B models: Llama2-13B and OPT-13B. SensZOQ still maintains its superior performance over both ICL and ZO Full FT. In the last row of OPT-13B (Table 4b), we ablate on the effect of 4-bit SqueezeLLM quantization and as expected, the average test accuracy will increase a little but not too much (∼0.1) after we remove the quantization on the dense weight parts.

Table 4: Fine-tuning performance of different methods. In the first column, "Q" means the full model is quantized with 4-bit quantization method (SqueezeLLM (Kim et al., 2024)), and "ZO" means the model is fine-tuned with ZO-SGD optimizer. For each cell, we report the mean and standard deviation of test set accuracy (↑) of 3 random trials in the format of mean_{std}. **We finally report the average accuracy across tasks in the last column.**

(a) **Llama2-13B**

	Methods	SST-2	RTE	CB	BoolQ	WSC	WiC	COPA	WinoG	**Avg**
Q, ZO	**SensZOQ**	$94.8_{0.4}$	$\mathbf{76.5}_{2.9}$	$\mathbf{86.3}_{2.2}$	$\mathbf{80.7}_{1.1}$	$\mathbf{56.7}_{1.6}$	$58.0_{2.4}$	$88.7_{0.9}$	$\mathbf{72.9}_{1.3}$	**76.8**
ZO	Full FT	$\mathbf{95.0}_{0.2}$	$74.4_{1.0}$	$85.7_{1.5}$	$80.5_{0.4}$	$54.8_{4.7}$	$\mathbf{60.9}_{1.4}$	$\mathbf{90.7}_{0.5}$	$70.5_{0.2}$	76.6
	Zero-shot	$30.4_{0.0}$	$50.9_{0.0}$	$44.6_{0.0}$	$57.0_{1.2}$	$41.3_{0.0}$	$50.8_{0.0}$	$83.0_{0.0}$	$65.2_{0.4}$	52.9
	ICL	$38.8_{4.8}$	$51.4_{1.9}$	$22.6_{6.1}$	$59.9_{3.0}$	$36.5_{0.0}$	$52.1_{0.9}$	$88.7_{1.7}$	$71.0_{0.5}$	52.6

(b) **OPT-13B**

Q, ZO	**SensZOQ**	$93.8_{0.4}$	$\mathbf{76.7}_{1.6}$	$65.5_{1.7}$	$\mathbf{72.8}_{0.6}$	$59.9_{0.5}$	$\mathbf{59.9}_{1.4}$	$\mathbf{88.7}_{1.2}$	$\mathbf{63.7}_{0.4}$	**72.6**
ZO	Full FT	$\mathbf{93.9}_{0.5}$	$74.0_{1.0}$	$\mathbf{67.9}_{2.5}$	$72.4_{0.3}$	$\mathbf{61.5}_{2.4}$	$58.6_{2.3}$	$87.0_{1.4}$	$63.3_{1.3}$	72.3
	Zero-shot	$61.0_{0.0}$	$58.5_{0.0}$	$48.2_{0.0}$	$59.8_{0.1}$	$36.5_{0.0}$	$52.0_{0.0}$	$80.0_{0.0}$	$60.7_{0.2}$	57.1
	ICL	$83.0_{8.5}$	$59.8_{4.2}$	$72.0_{1.7}$	$71.6_{2.4}$	$38.1_{2.3}$	$53.6_{2.2}$	$84.0_{2.9}$	$63.2_{0.8}$	65.6
ZO	Sens. (C4, static)	$93.3_{0.2}$	$75.2_{1.0}$	$64.2_{1.5}$	$73.5_{0.3}$	$62.5_{2.1}$	$60.0_{1.7}$	$87.3_{0.5}$	$65.5_{0.7}$	72.7

Commonsense reasoning, math, MMLU dataset results. In Table 5, we still compare SensZOQ vs. ZO full FT & ICL in standard commonsense benchmarks (Hu et al., 2023; Yang et al., 2024), 1 math algebraic word problem task AQuA (Ling et al., 2017), and MMLU (Hendrycks et al., 2021). **To the best of our knowledge, there are no ZO-LLM research yet evaluated on harder commonsense reasoning or math tasks. We take a pioneering step in this direction and establish the ZO baselines.**

SensZOQ still achieves the highest average accuracy. If we compare SensZOQ with ZO full FT on pairs, SensZOQ wins 5/8 for commonsense reasoning, and the math task. However, SensZOQ loses in MMLU task, and we speculate this might be due to a data distribution mismatch between C4 and education domain/expert-level QA in MMLU. In Table 6 we find switching the source of extracting sensitive parameters from C4 to OpenWebMath or ArXiv will significantly close this gap. Even so, C4 is still a *generally good* choice.

Table 5: Fine-tuning performance of different methods for **Mistral-7B** on 8 commonsense reasoning tasks (*cs*), 1 math task (*math*), and MMLU task. For each cell, we report the mean and standard deviation of test set accuracy (↑) of 3 random trials in the format of mean_{std}. **We finally report the average accuracy across tasks in the last column.**

Methods	Arc-E (*cs*)	Arc-C (*cs*)	HS (*cs*)	OBQA (*cs*)	PIQA (*cs*)	SIQA (*cs*)	BoolQ (*cs*)	WinoG (*cs*)	AQuA (*math*)	MMLU	**Avg**
SensZOQ	$88.6_{0.1}$	$77.9_{1.3}$	$\mathbf{82.1}_{0.4}$	$76.5_{0.6}$	$\mathbf{84.5}_{0.5}$	$68.1_{0.4}$	$75.1_{2.4}$	$\mathbf{74.1}_{0.5}$	$\mathbf{27.7}_{2.1}$	$58.2_{0.2}$	**71.3**
ZO Full FT	$89.2_{0.5}$	$78.6_{0.8}$	$80.5_{1.1}$	$76.4_{0.3}$	$84.1_{0.2}$	$67.6_{0.1}$	$\mathbf{76.6}_{0.2}$	$72.2_{0.5}$	$24.1_{2.0}$	$\mathbf{59.2}_{0.1}$	70.9
Zero-shot	$86.8_{0.0}$	$75.9_{0.0}$	$77.9_{0.8}$	$71.0_{0.0}$	$82.1_{0.3}$	$59.9_{0.5}$	$43.4_{1.8}$	$66.2_{0.1}$	$23.5_{1.9}$	$57.5_{0.0}$	64.4
ICL	$\mathbf{90.5}_{0.2}$	$\mathbf{80.0}_{2.0}$	$80.3_{1.4}$	$\mathbf{79.8}_{0.7}$	$\mathbf{84.5}_{0.9}$	$\mathbf{69.9}_{1.0}$	$46.8_{6.5}$	$74.0_{0.8}$	$26.6_{1.1}$	$\mathbf{59.2}_{0.2}$	69.2

Alternative data sources for extracting sensitive parameters. C4 (Raffel et al., 2019) is cleaned from the CommonCrawl and covers a wide range of domains, and this drives us to adopt it as the default choice for extracting sensitive parameters. Here we evaluate the SensZOQ's performance when fine-tuning sensitive parameters extracted from some alternative text choices that have certain domain specialties. As a case study, we only pick 3 alternative choices commonly selected for pretraining dataset mixtures (e.g. Dolma (Soldaini et al., 2024)) as follows:

- OpenWebMath (Paster et al., 2024), a pile of Internet mathematical proofs. `https://huggingface.co/datasets/open-web-math/open-web-math`
- ArXiv (Cohan et al., 2018), a pile of scientific papers. We use the ArXiv articles subset from this dataset. `https://huggingface.co/datasets/armanc/scientific_papers`
- Wiki103 (Merity et al., 2016), a pile of selected Wikipedia articles. `https://huggingface.co/datasets/Salesforce/wikitext`

Table 6: Fine-tuning performance of SensZOQ with **0.1% sensitive parameters extracted from C4, OpenWebMath, ArXiv, and Wiki103** for **Mistral-7B** on 8 commonsense reasoning tasks, 1 math task, MMLU task, and 3 SuperGLUE tasks (task category follows this order and is separated by the vertical bar). For each cell, we report the mean accuracy (↑) over 3 random trials. In the last row, we give ZO Full FT & ICL baselines as reference. **We finally report the average accuracy across tasks in the last column.**

Source	Arc-E	Arc-C	HS	OBQA	PIQA	SIQA	BoolQ	WinoG	AQuA	MMLU	RTE	WiC	COPA	**Avg**
C4	**88.6**	77.9	**82.1**	**76.5**	**84.5**	68.1	75.1	**74.1**	**27.7**	58.2	**78.0**	**63.6**	88.3	**72.5**
OpenWebMath	87.8	**78.5**	81.8	74.1	83.7	67.2	71.5	72.2	25.2	**58.8**	66.7	60.8	**89.0**	70.6
ArXiv	87.7	77.0	82.7	75.2	84.2	**68.7**	69.1	72.2	25.9	**58.8**	70.0	59.4	**89.0**	70.8
Wiki103	87.8	77.9	82.0	73.0	83.9	68.6	**79.7**	73.2	26.4	57.6	69.2	60.9	88.7	71.5
ZO Full FT	89.2	78.6	80.5	76.4	84.1	67.6	76.6	72.2	24.1	59.2	74.6	62.6	88.3	71.8
ICL	90.5	80.0	80.3	79.8	84.5	69.9	46.8	74.0	26.6	59.2	55.2	63.8	74.0	68.0

In Table 6, we can find that when finetuning 0.1% sensitive parameters with Mistral-7B, **C4 achieves the highest average accuracy**, with notable performance on commonsense QA tasks like OBQA and PIQA, and NLU tasks like RTE and WiC. If we want better performance on education or hard reasoning tasks like Arc-C or expert-level QA task like MMLU, OpenWebMath and ArXiv is a better choice than C4.

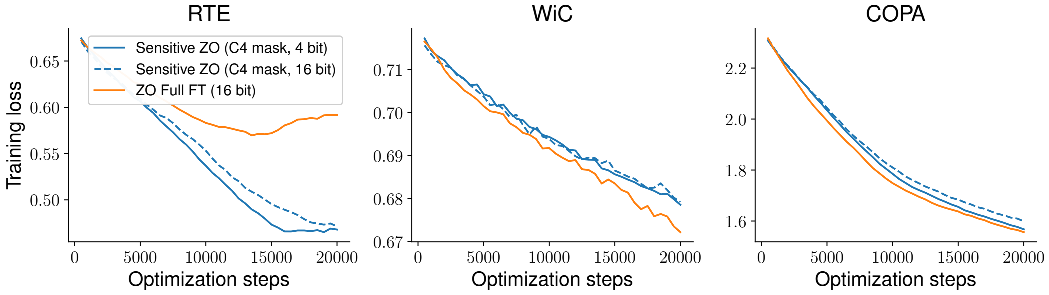
We believe that C4 is not the *only* choice but rather a *generally good* choice for downstream tasks, and this is quite important as we are using the same set of sparse parameters for different tasks and we want it to yield satisfactory performance for as many tasks as possible. Otherwise we will have to create a separate sparse mask and quantized models for each task and this will make our method really unpractical.

E Empirical convergence of SensZOQ

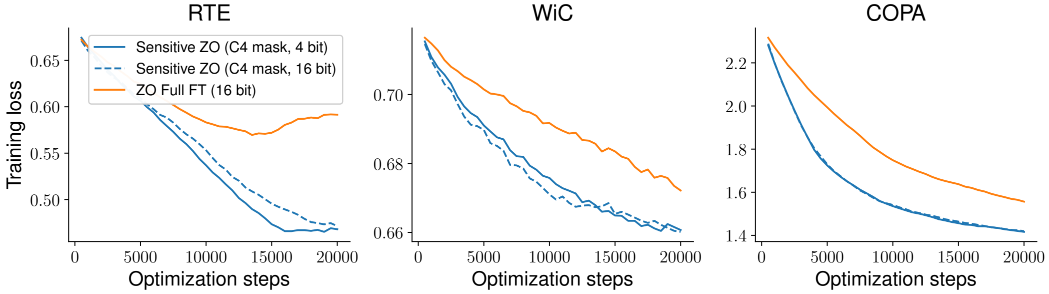
To validate the faster convergence rate of sensitive sparse ZO-SGD proven in Theorem 1, we plot the training loss of SensZOQ vs. ZO Full FT w.r.t. 20k optimization steps for OPT-13B fine-tuning tasks, as shown in Figure 6. Note that the "Sensitive ZO (C4 mask, 4 bit)" is our SensZOQ method, and the Sensitive ZO (C4 mask, 16 bit) will keep the dense weights in FP16 as it more aligns with our theory.

1. In Figure 6a, we find that **an in-place learning rate transfer from ZO Full FT to SensZOQcan already result in similar or faster convergence.**
2. In Figure 6b, **we identify that SensZOQ can tolerate higher learning rate without divergence.** For example, the best learning rates for ZO Full FT in WiC and COPA are 2e-7 (5e-7 would result in a divergence) while SensZOQ can tolerate 5e-7 and produce much faster convergence results.

In conclusion, SensZOQ would have at least similar convergence as ZO Full FT with the same learning rate (so a direct learning rate transfer would already yield satisfactory results), and with the same learning rate grid search, SensZOQ can achieve much faster convergence in practice.



(a) Convergence of Sensitive ZO vs. ZO Full FT under the best learning rate for ZO Full FT only.



(b) Convergence of Sensitive ZO vs. ZO Full FT under the best learning rate for each method.

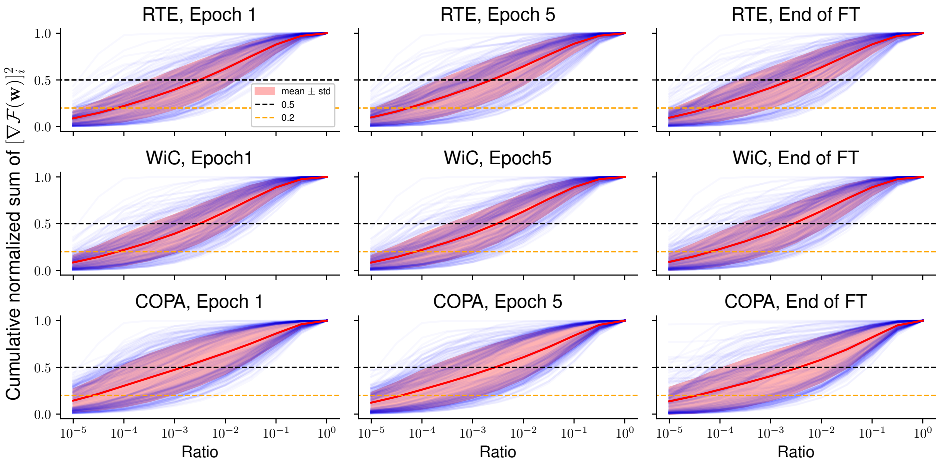
Figure 6: Convergence of SensZOQ for OPT-13B across 3 fine-tuning tasks. SensZOQ corresponds to "Sensitive ZO (C4 mask, 4 bit)", and we also provide the convergence results of unquantized weight (16 bit) as a reference. In Figure 6a, We first search the best learning rate for ZO Full FT that reaches the lowest training loss in [1e-7, 2e-7, 5e-7, 1e-6] grid, and we use such learning rate for SensZOQ. In Figure 6b, we search for the best learning rate for ZO Full FT and SensZOQ separately. The other hyperparameters (perturbation constant ϵ and minibatch size B) are kept the same in both Figure 6a and Figure 6b experiments.

F SENSITIVE PARAMETERS IN LLMS UNDER MICROSCOPE

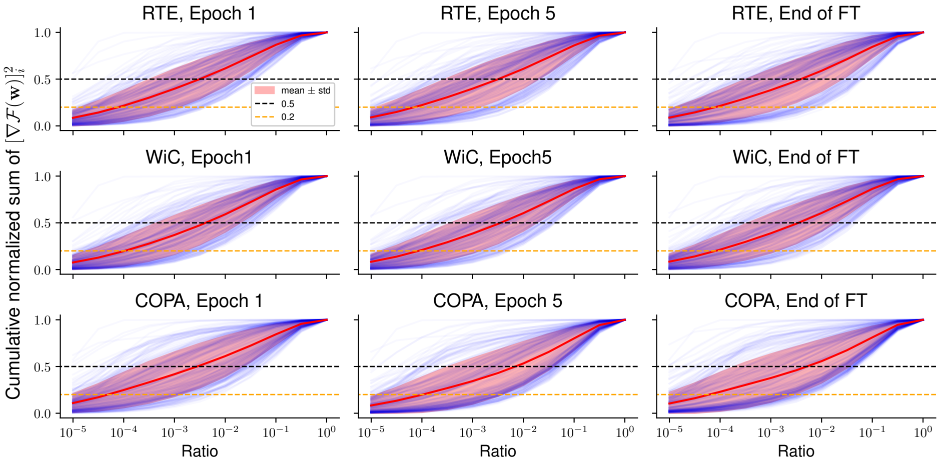
F.1 GRADIENT SPARSITY DURING LLM FINE-TUNING

SuperGLUE tasks. In Figure 2, 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-SGD gradient sparsity for Llama2-7B, Mistral-7B, and OPT-6.7B during epoch 1, 5, and 10 (end of fine-tuning).

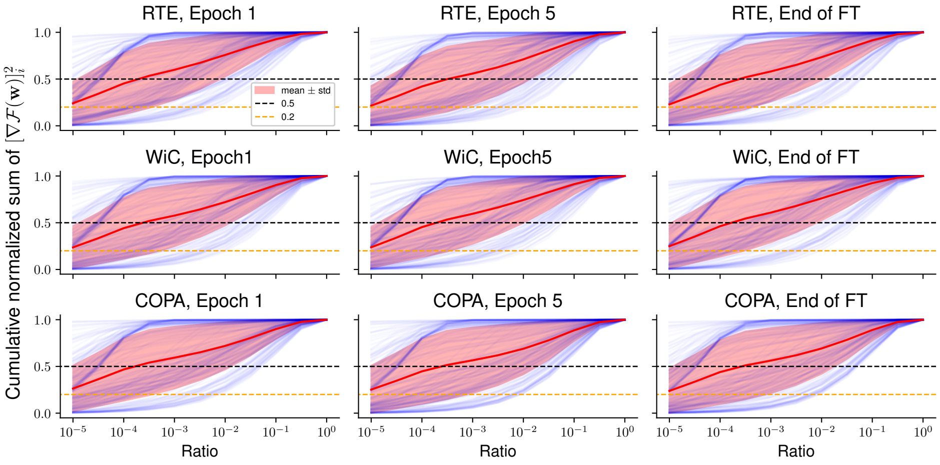
We observe that the gradient sparsity is exhibited throughout the fine-tuning with slightly increasing towards the end. OPT-6.7B which uses ReLU as the activation function would demonstrate greater sparsity across tasks compared with Llama2-7B and Mistral-7B which uses SwiGLU and SiLU respectively. Nevertheless, 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 7: Cumulative normalized sum of coordinate-wise gradient square $[\nabla\mathcal{F}(\mathbf{w})]_i^2$ of linear layers for Llama2-7B (subfigure 7a), Mistral-7B (subfigure 7b), and OPT-6.7B (subfigure 7c) across RTE, WiC, and COPA tasks during FO-SGD full FT for 10 epochs. 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.

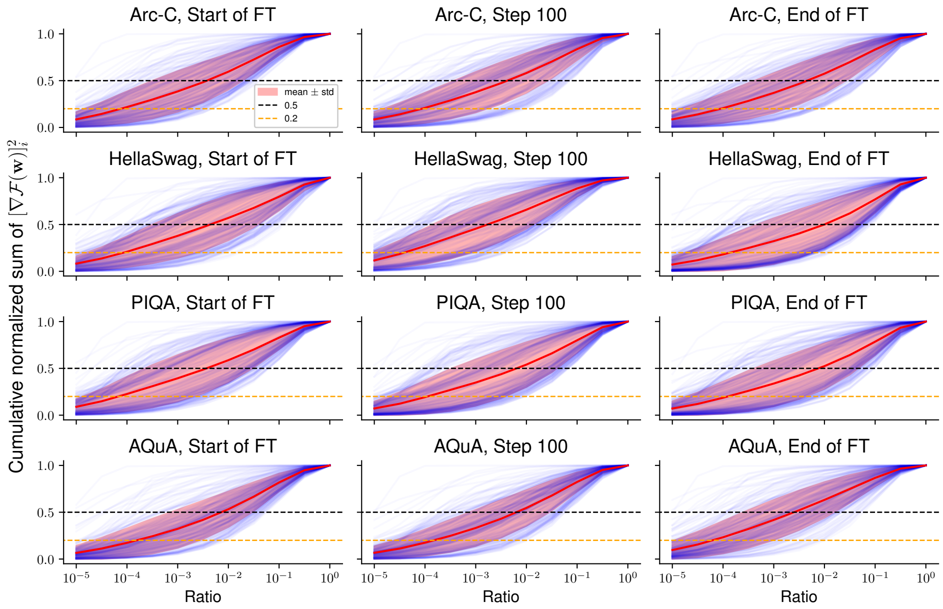


Figure 8: Cumulative normalized sum of coordinate-wise gradient square $[\nabla\mathcal{F}(\mathbf{w})]_i^2$ of linear layers for **Mistral-7B** across Arc-C, HS, PIQA, and AQuA tasks during FO-Adam full FT for 1000 steps.

Reasoning tasks. We further analyze the sparsity of sensitive parameters for hard reasoning tasks such as Arc-C, HellaSwag, PIQA, and math reasoning task AQuA. We follow the same methodology that produced Figure 7b to produce Figure 8. We can see that sensitive parameters are still fairly sparse for these 4 tasks. Specifically,

- For 3 commonsense reasoning tasks, the argument that 0.1% of parameters contribute about 50% gradient norm still holds.
- For the math task, the sensitive parameters tend to be slightly denser, but 1% of parameters can still cover 50% gradient norm (0.1% will cover ∼30% gradient norm).

F.2 BLOCK-WISE AND LAYER-WISE GRADIENT SPARSITY

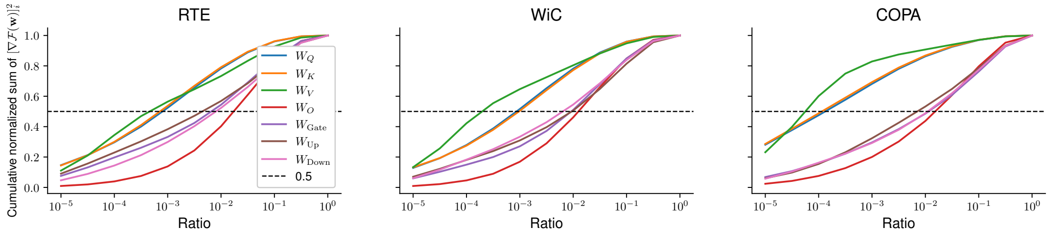
Although this is not the main focus of this paper, we also inspect the gradient sparsity for each weight type $\{W_Q, W_K, W_V, W_O, (W_{\text{Gate}}), W_{\text{Up}}, W_{\text{Down}}\}$ and layers $\{1, 8, 16, 24, 32\}$. This might inspire future adaptive sensitive parameter selection based on the weight type or model depth in different transformers. The results for weight type are shown in Figure 9 and for layers are shown in Figure 10. We order the gradient sparsity levels from top to down and get the following rankings across the blocks and layers (the higher-ranked the sparser) as the following 2 lists.

Weight types. It is consistent across all 3 models that W_V has the highest gradient sparsity and W_O has the lowest gradient sparsity.

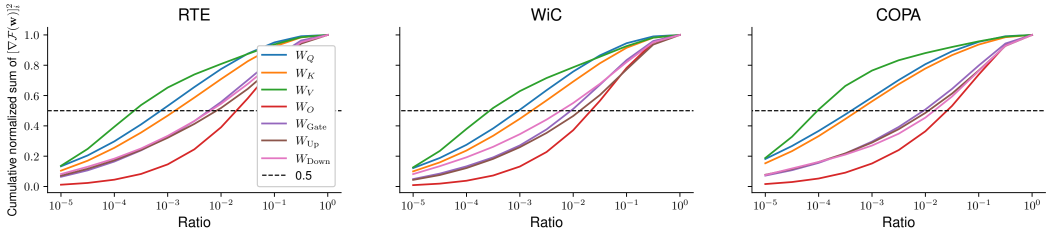
- **Llama2-7B** (Figure 9a): $W_V > W_Q \sim W_K > W_{\text{Gate}} \sim W_{\text{Up}} \sim W_{\text{Down}} > W_O$.
- **Mistral-7B** (Figure 9b): $W_V > W_Q \sim W_K > W_{\text{Gate}} \sim W_{\text{Up}} \sim W_{\text{Down}} > W_O$.
- **OPT-6.7B** (Figure 9c): $W_V > W_{\text{Up}} > W_Q > W_{\text{Down}} \sim W_K > W_O$.

Layer depth. For Llama2-7B and Mistral-7B, it is consistent that Layer 1 (first layer) has the highest gradient sparsity followed by Layer 32 (last layer). The gradient sparsity for middle layers are lower and do not show a meaningful trend. The only meaningful result for OPT-6.7B is that Layer 1 has the lowest gradient sparsity.

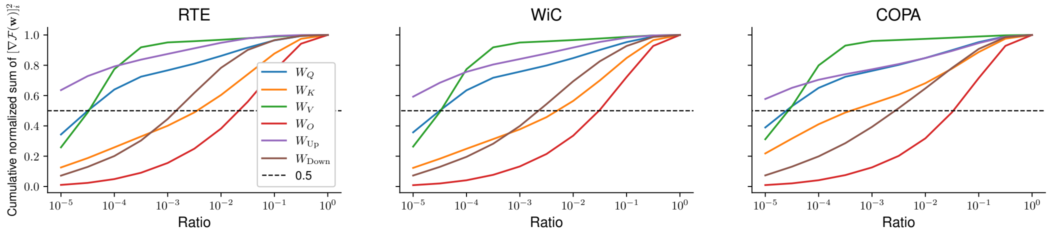
- **Llama2-7B** (Figure 10a): Layer 1 > Layer 32 ∼ Layer 24 > Layer 8 ∼ Layer 16
- **Mistral-7B** (Figure 10b): Layer 1 > Layer 32 > Layer 8 ∼ Layer 16 ∼ Layer 24
- **OPT-6.7B** (Figure 10c): Layer 8 ∼ Layer 16 > Layer 24 ∼ Layer 32 > Layer 1



(a) Llama2-7B



(b) Mistral-7B



(c) OPT-6.7B

Figure 9: Cumulative normalized sum of coordinate-wise $[\nabla\mathcal{F}(w)]_i^2$ of Llama2-7B (subfigure 9a), Mistral-7B (subfigure 9b), and OPT-6.7B (subfigure 9c)'s weight blocks at Epoch 1 of FO-SGD full fine-tuning. For a given model and a given weight block type, we report the average value across 32 transformer layers as a line in each subfigure.

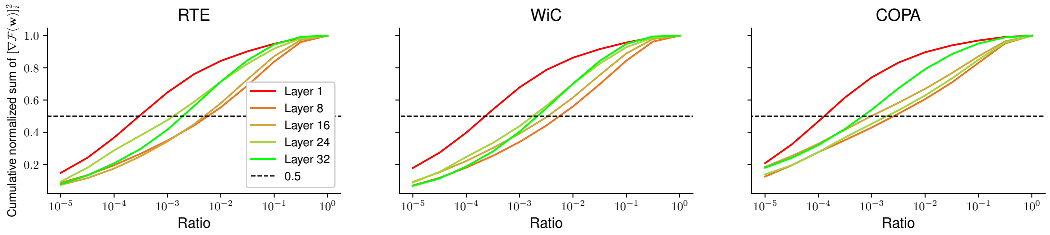
F.3 TRANSFERABILITY OF GRADIENT FEATURES FROM PRE-TRAINING DATASETS TO DOWNSTREAM TASKS

SuperGLUE tasks. In Figure 3, we explore the transferability of gradient features from pre-training datasets (C4) to downstream tasks, and here we will also validate this phenomenon across models, as shown in Figure 11. 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. As a comparison, we also include the results of weights with largest magnitude, weights with smallest magnitude, and random subsets in Figure 12 at 1e-3 nnz threshold (same as 1e-3 in Figure 11) for comparison. It is clear that "C4 grad (static)" has exponentially higher similarity with "task grad (dyn.)" than all of these 3 baselines. *We also note that weights with largest magnitude (weight outliers) are usually NOT gradient outliers as observed in Figure 12.*

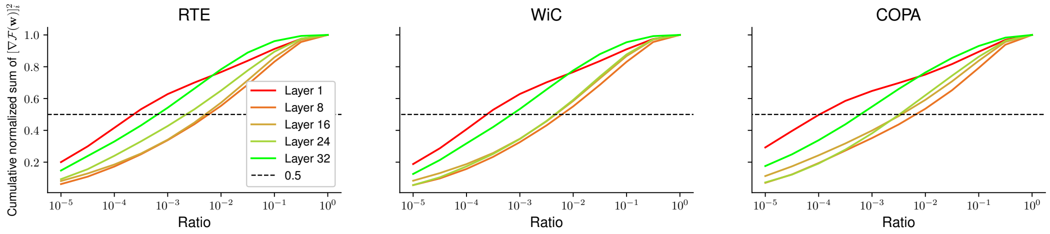
In this case, sensitive parameters determined from C4 gradients would still be similar to sensitive parameters determined from downstream task-specific gradients across models.

Reasoning tasks. The transferability results for reasoning tasks are shown in Figure 13.

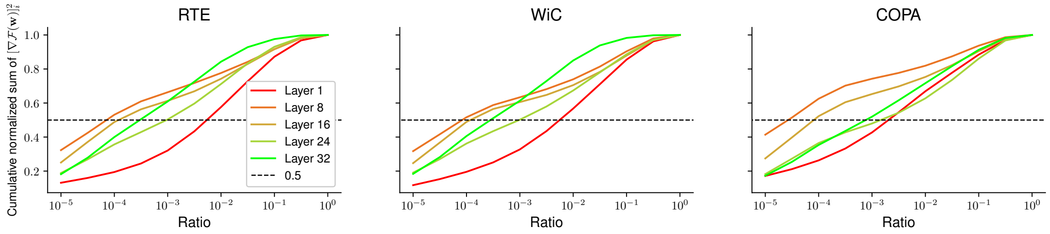
As there are still no solid lines (for all top-(1e-2,1e-3,1e-4) parameters with C4 mask) vanish to 0, C4 gradient mask still demonstrates great transferability to these 3 commonsense and 1 math task. For top-1e-3 C4 gradient mask, the lowest covered gradient norm is ∼0.2, while the maximum possible (*task grad, dyn.*) is ∼0.6 across tasks.



(a) Llama2-7B



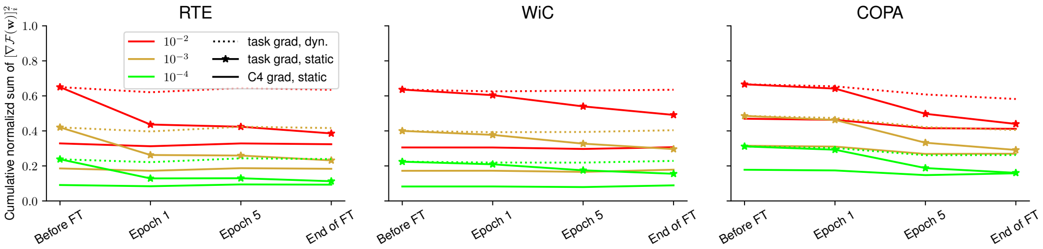
(b) Mistral-7B



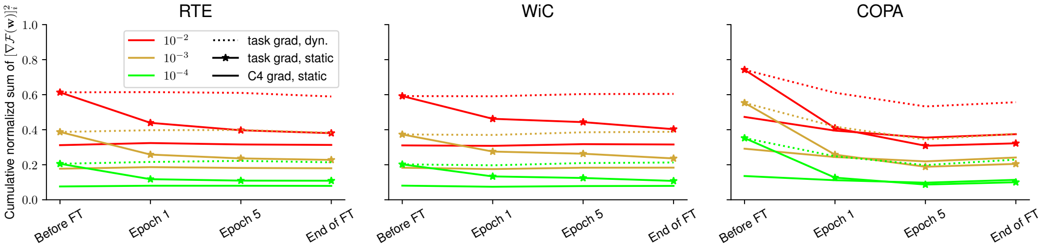
(c) OPT-6.7B

Figure 10: Cumulative normalized sum of coordinate-wise $[\nabla\mathcal{F}(w)]_i^2$ of Llama2-7B (subfigure 10a), Mistral-7B (subfigure 10b), and OPT-6.7B (subfigure 10c)'s linear weights at Epoch 1 of FO-SGD full fine-tuning. For a given model and a given transformer layer, we report the average value across all linear layers at this transformer layer as a line in each subfigure.

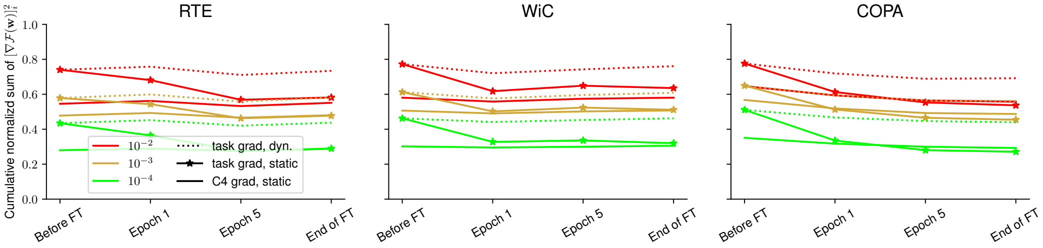
We still note that for AQuA (math algebraic word problem task), C4's transferability is weaker than the other 3 commonsense reasoning tasks. We speculate that this is due to the need to learn more math-related knowledge during FT as the covered task gradient squares by task gradient mask before FT mask (*task grad, static*) also declines more during FT.



(a) Llama2-7B

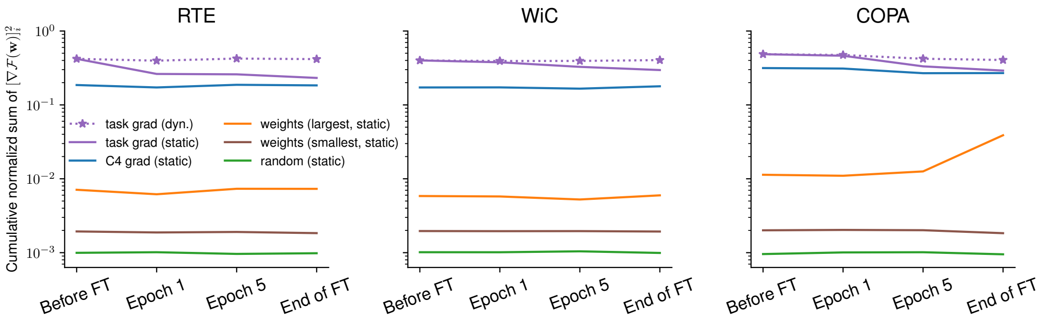


(b) Mistral-7B

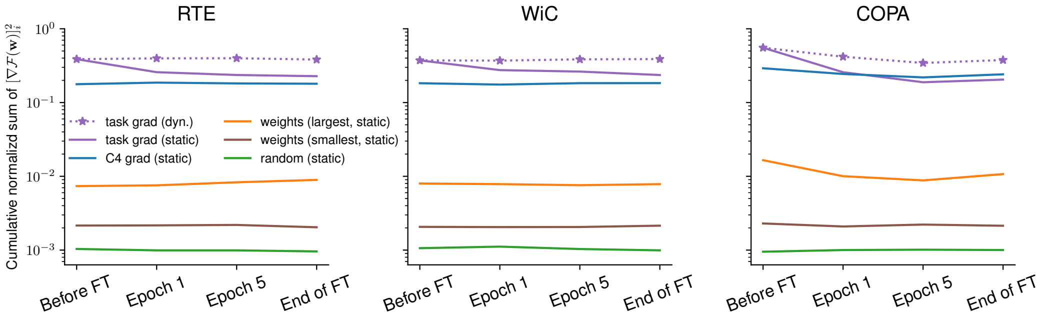


(c) OPT-6.7B

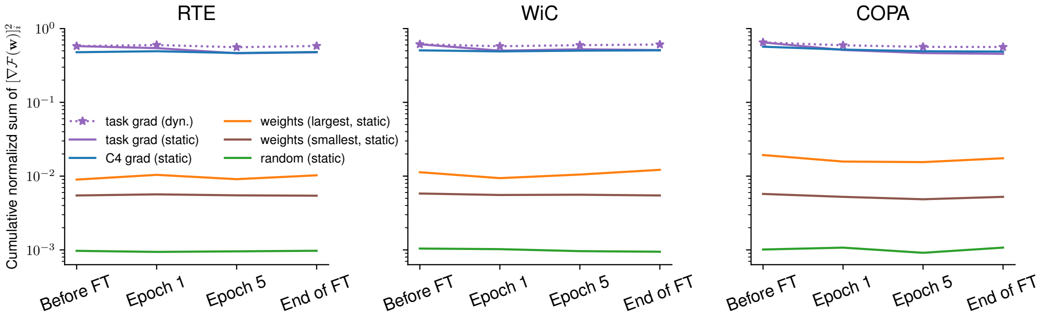
Figure 11: Cumulative normalized sum of coordinate-wise $[\nabla \mathcal{F}(w)]_i^2$ of Llama2-7B (subfigure 11a), Mistral-7B (subfigure 11b), and OPT-6.7B (subfigure 11c)'s linear layers **after applying sparsity masks of each method** during FO-SGD full fine-tuning for 10 epochs. For a given model 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.



(a) Llama2-7B



(b) Mistral-7B



(c) OPT-6.7B

Figure 12: Cumulative normalized sum of coordinate-wise $[\nabla \mathcal{F}(w)]_i^2$ of Llama2-7B (subfigure 12a), Mistral-7B (subfigure 12b), and OPT-6.7B (subfigure 12c)'s linear layers **after applying 99.9% sparsity masks of each method** during FO-SGD full fine-tuning. The results of "C4 grad, static", "task grad, dyn.", and "task grad, static" are the same as their 1e-3 results in Figure 11.

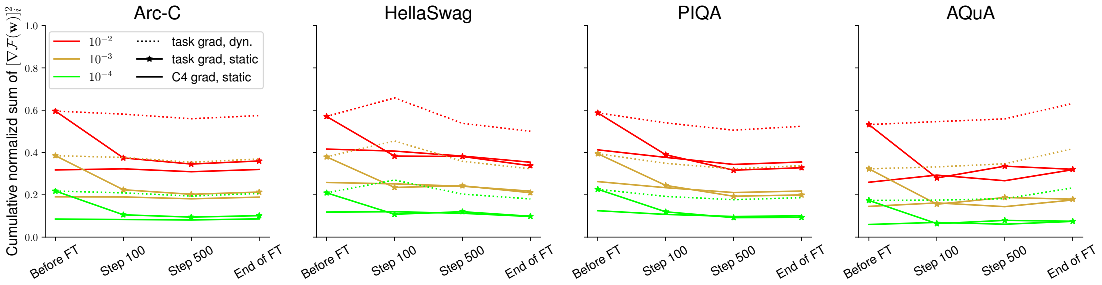


Figure 13: Cumulative normalized sum of coordinate-wise $[\nabla \mathcal{F}(w)]_i^2$ of Mistral-7B after applying sparsity masks of each method during FO-Adam full FT for 1000 steps.

F.4 OVERLAP RATIO OF TOP GRADIENT FEATURES

Besides Table 6, we dive deeper into the overlap ratio of top gradient entries from different data sources as shown in Figure 14.

Notice that the order here maps to the column/row rank for each subfigure in Figure 14.

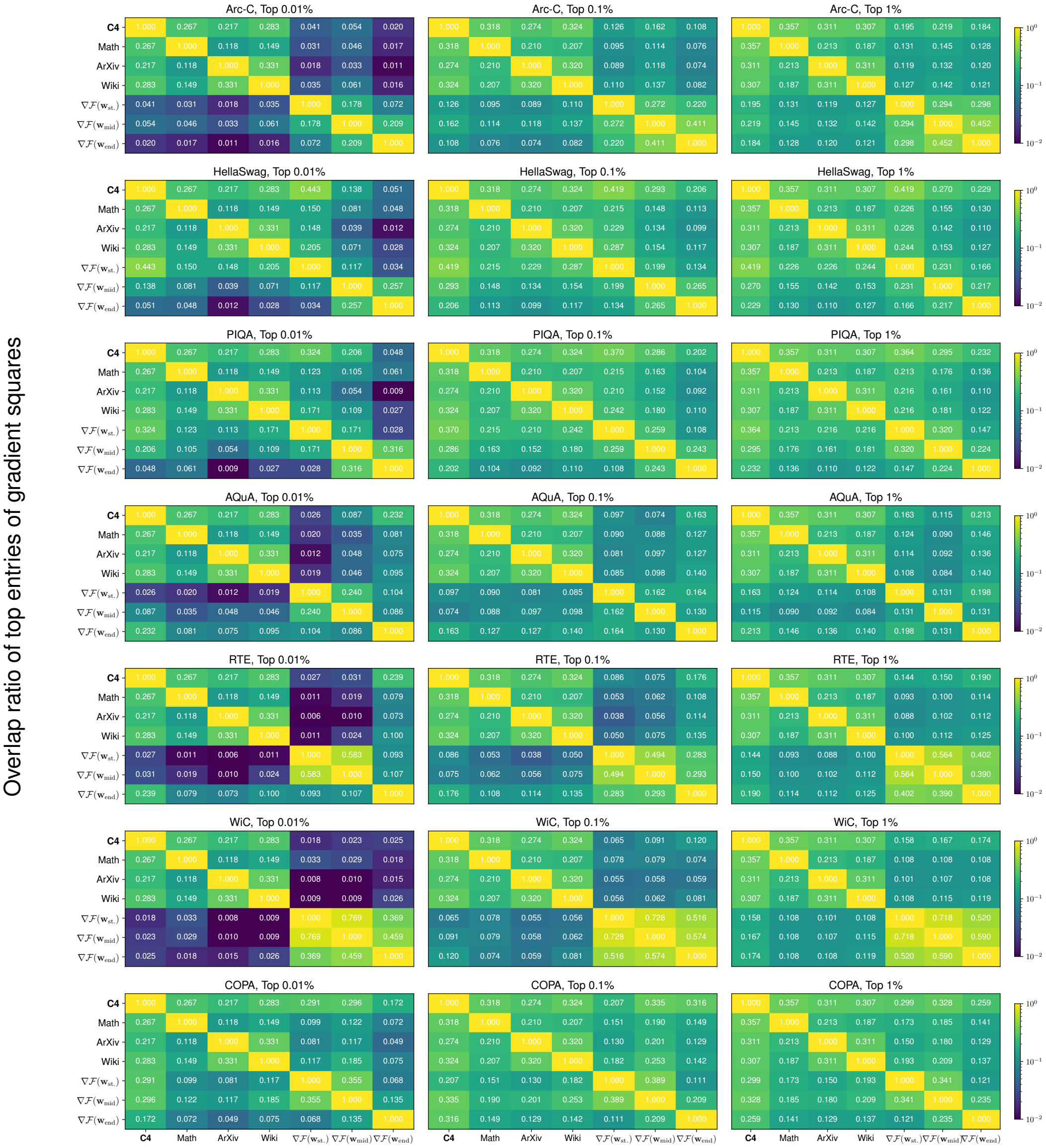
1. **C4**, SensZOQ's choice.
2. OpenWebMath
3. ArXiv
4. Wiki103
5. Task gradient before FT (start). We abbreviate it as $\nabla\mathcal{F}(\mathbf{w}_{\text{st}})$.
6. Task gradient after 10% finetuning steps (mid). We abbreviate it as $\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$.
7. Task gradient at the end of FT (end). We abbreviate it as $\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$.

We consider 7 tasks (3 commonsense reasoning tasks *Arc-C*, *HellaSwag*, *PIQA*, 1 math task *AQuA*, 3 SuperGLUE tasks *RTE*, *WiC*, *COPA*).

For a quick overview, we include Figure 15 as the average overlap ratio across tasks for the 4 pretraining text. The second row in Figure 15 gives an empirical evidence for the "fixed gradient feature" during FT as the top entries in $\nabla\mathcal{F}(\mathbf{w}_{\text{st}})$ (task grad before FT) resemble $\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$ (task grad during FT) and $\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$ (task grad during FT)

Empirical findings

- The top gradient entries from all 4 pretraining text corpus still overlap considerably with the task gradient, at least for top-0.1% entries.
- C4 generally covers more top gradient entries than OpenWebMath, ArXiv, and Wiki103.



Arc-C, Top 0.01%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.267	0.217	0.283	0.041	0.054	0.020
Math	0.267	1.000	0.118	0.149	0.031	0.046	0.017
ArXiv	0.217	0.118	1.000	0.331	0.018	0.033	0.011
Wiki	0.283	0.149	0.331	1.000	0.035	0.061	0.016
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.041	0.031	0.018	0.035	1.000	0.178	0.072
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.054	0.046	0.033	0.061	0.178	1.000	0.209
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.020	0.017	0.011	0.016	0.072	0.209	1.000

Arc-C, Top 0.1%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.318	0.274	0.324	0.126	0.162	0.108
Math	0.318	1.000	0.210	0.207	0.095	0.114	0.076
ArXiv	0.274	0.210	1.000	0.320	0.089	0.118	0.074
Wiki	0.324	0.207	0.320	1.000	0.110	0.137	0.082
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.126	0.095	0.089	0.110	1.000	0.272	0.220
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.162	0.114	0.118	0.137	0.272	1.000	0.411
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.108	0.076	0.074	0.082	0.220	0.411	1.000

Arc-C, Top 1%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.357	0.311	0.307	0.195	0.219	0.184
Math	0.357	1.000	0.213	0.187	0.131	0.145	0.128
ArXiv	0.311	0.213	1.000	0.311	0.119	0.132	0.120
Wiki	0.307	0.187	0.311	1.000	0.127	0.142	0.121
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.195	0.131	0.119	0.127	1.000	0.294	0.298
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.219	0.145	0.132	0.142	0.294	1.000	0.452
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.184	0.128	0.120	0.121	0.298	0.452	1.000

HellaSwag, Top 0.01%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.267	0.217	0.283	0.443	0.138	0.051
Math	0.267	1.000	0.118	0.149	0.150	0.081	0.048
ArXiv	0.217	0.118	1.000	0.331	0.148	0.039	0.012
Wiki	0.283	0.149	0.331	1.000	0.205	0.071	0.028
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.443	0.150	0.148	0.205	1.000	0.117	0.034
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.138	0.081	0.039	0.071	0.117	1.000	0.257
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.051	0.048	0.012	0.028	0.034	0.257	1.000

HellaSwag, Top 0.1%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.318	0.274	0.324	0.419	0.293	0.206
Math	0.318	1.000	0.210	0.207	0.215	0.148	0.113
ArXiv	0.274	0.210	1.000	0.320	0.229	0.134	0.099
Wiki	0.324	0.207	0.320	1.000	0.287	0.154	0.117
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.419	0.215	0.229	0.287	1.000	0.199	0.134
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.293	0.148	0.134	0.154	0.199	1.000	0.265
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.206	0.113	0.099	0.117	0.134	0.265	1.000

HellaSwag, Top 1%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.357	0.311	0.307	0.419	0.270	0.229
Math	0.357	1.000	0.213	0.187	0.226	0.155	0.130
ArXiv	0.311	0.213	1.000	0.311	0.226	0.142	0.110
Wiki	0.307	0.187	0.311	1.000	0.244	0.153	0.127
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.419	0.226	0.226	0.244	1.000	0.231	0.166
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.270	0.155	0.142	0.153	0.231	1.000	0.217
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.229	0.130	0.110	0.127	0.166	0.217	1.000

PIQA, Top 0.01%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.267	0.217	0.283	0.324	0.206	0.048
Math	0.267	1.000	0.118	0.149	0.123	0.105	0.061
ArXiv	0.217	0.118	1.000	0.331	0.113	0.054	0.009
Wiki	0.283	0.149	0.331	1.000	0.171	0.109	0.027
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.324	0.123	0.113	0.171	1.000	0.171	0.028
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.206	0.105	0.054	0.109	0.171	1.000	0.316
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.048	0.061	0.009	0.027	0.028	0.316	1.000

PIQA, Top 0.1%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.318	0.274	0.324	0.370	0.286	0.202
Math	0.318	1.000	0.210	0.207	0.215	0.163	0.104
ArXiv	0.274	0.210	1.000	0.320	0.210	0.152	0.092
Wiki	0.324	0.207	0.320	1.000	0.242	0.180	0.110
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.370	0.215	0.210	0.242	1.000	0.259	0.108
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.286	0.163	0.152	0.180	0.259	1.000	0.243
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.202	0.104	0.092	0.110	0.108	0.243	1.000

PIQA, Top 1%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.357	0.311	0.307	0.364	0.295	0.232
Math	0.357	1.000	0.213	0.187	0.213	0.176	0.136
ArXiv	0.311	0.213	1.000	0.311	0.216	0.161	0.110
Wiki	0.307	0.187	0.311	1.000	0.216	0.181	0.122
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.364	0.213	0.216	0.216	1.000	0.320	0.147
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.295	0.176	0.161	0.181	0.320	1.000	0.224
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.232	0.136	0.110	0.122	0.147	0.224	1.000

AQuA, Top 0.01%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.267	0.217	0.283	0.026	0.087	0.232
Math	0.267	1.000	0.118	0.149	0.020	0.035	0.081
ArXiv	0.217	0.118	1.000	0.331	0.012	0.048	0.075
Wiki	0.283	0.149	0.331	1.000	0.019	0.046	0.095
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.026	0.020	0.012	0.019	1.000	0.240	0.104
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.087	0.035	0.048	0.046	0.240	1.000	0.086
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.232	0.081	0.075	0.095	0.104	0.086	1.000

AQuA, Top 0.1%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.318	0.274	0.324	0.097	0.074	0.163
Math	0.318	1.000	0.210	0.207	0.090	0.088	0.127
ArXiv	0.274	0.210	1.000	0.320	0.081	0.097	0.127
Wiki	0.324	0.207	0.320	1.000	0.085	0.098	0.140
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.097	0.090	0.081	0.085	1.000	0.162	0.164
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.074	0.088	0.097	0.098	0.162	1.000	0.130
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.163	0.127	0.127	0.140	0.164	0.130	1.000

AQuA, Top 1%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.357	0.311	0.307	0.163	0.115	0.213
Math	0.357	1.000	0.213	0.187	0.124	0.090	0.146
ArXiv	0.311	0.213	1.000	0.311	0.114	0.092	0.136
Wiki	0.307	0.187	0.311	1.000	0.108	0.084	0.140
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.163	0.124	0.114	0.108	1.000	0.131	0.198
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.115	0.090	0.092	0.084	0.131	1.000	0.131
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.213	0.146	0.136	0.140	0.198	0.131	1.000

RTE, Top 0.01%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.267	0.217	0.283	0.027	0.031	0.239
Math	0.267	1.000	0.118	0.149	0.011	0.019	0.079
ArXiv	0.217	0.118	1.000	0.331	0.006	0.010	0.073
Wiki	0.283	0.149	0.331	1.000	0.011	0.024	0.100
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.027	0.011	0.006	0.011	1.000	0.583	0.093
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.031	0.019	0.010	0.024	0.583	1.000	0.107
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.239	0.079	0.073	0.100	0.093	0.107	1.000

RTE, Top 0.1%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.318	0.274	0.324	0.086	0.075	0.176
Math	0.318	1.000	0.210	0.207	0.053	0.062	0.108
ArXiv	0.274	0.210	1.000	0.320	0.038	0.056	0.114
Wiki	0.324	0.207	0.320	1.000	0.050	0.075	0.135
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.086	0.053	0.038	0.050	1.000	0.494	0.283
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.075	0.062	0.056	0.075	0.494	1.000	0.293
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.176	0.108	0.114	0.135	0.283	0.293	1.000

RTE, Top 1%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.357	0.311	0.307	0.144	0.150	0.190
Math	0.357	1.000	0.213	0.187	0.093	0.100	0.114
ArXiv	0.311	0.213	1.000	0.311	0.088	0.102	0.112
Wiki	0.307	0.187	0.311	1.000	0.100	0.112	0.125
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.144	0.093	0.088	0.100	1.000	0.564	0.402
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.150	0.100	0.102	0.112	0.564	1.000	0.390
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.190	0.114	0.112	0.125	0.402	0.390	1.000

WiC, Top 0.01%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.267	0.217	0.283	0.018	0.023	0.025
Math	0.267	1.000	0.118	0.149	0.033	0.029	0.018
ArXiv	0.217	0.118	1.000	0.331	0.008	0.010	0.015
Wiki	0.283	0.149	0.331	1.000	0.009	0.009	0.026
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.018	0.033	0.008	0.009	1.000	0.769	0.369
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.023	0.029	0.010	0.009	0.769	1.000	0.459
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.025	0.018	0.015	0.026	0.369	0.459	1.000

WiC, Top 0.1%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.318	0.274	0.324	0.065	0.091	0.120
Math	0.318	1.000	0.210	0.207	0.078	0.079	0.074
ArXiv	0.274	0.210	1.000	0.320	0.055	0.058	0.059
Wiki	0.324	0.207	0.320	1.000	0.056	0.062	0.081
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.065	0.078	0.055	0.056	1.000	0.728	0.516
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.091	0.079	0.058	0.062	0.728	1.000	0.574
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.120	0.074	0.059	0.081	0.516	0.574	1.000

WiC, Top 1%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.357	0.311	0.307	0.158	0.167	0.174
Math	0.357	1.000	0.213	0.187	0.108	0.108	0.108
ArXiv	0.311	0.213	1.000	0.311	0.101	0.107	0.108
Wiki	0.307	0.187	0.311	1.000	0.108	0.115	0.119
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.158	0.108	0.101	0.108	1.000	0.718	0.520
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.167	0.108	0.107	0.115	0.718	1.000	0.590
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.174	0.108	0.108	0.119	0.520	0.590	1.000

COPA, Top 0.01%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.267	0.217	0.283	0.291	0.296	0.172
Math	0.267	1.000	0.118	0.149	0.099	0.122	0.072
ArXiv	0.217	0.118	1.000	0.331	0.081	0.117	0.049
Wiki	0.283	0.149	0.331	1.000	0.117	0.185	0.075
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.291	0.099	0.081	0.117	1.000	0.355	0.068
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.296	0.122	0.117	0.185	0.355	1.000	0.135
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.172	0.072	0.049	0.075	0.068	0.135	1.000

COPA, Top 0.1%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.318	0.274	0.324	0.207	0.335	0.316
Math	0.318	1.000	0.210	0.207	0.151	0.190	0.149
ArXiv	0.274	0.210	1.000	0.320	0.130	0.201	0.129
Wiki	0.324	0.207	0.320	1.000	0.182	0.253	0.142
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.207	0.151	0.130	0.182	1.000	0.389	0.111
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.335	0.190	0.201	0.253	0.389	1.000	0.209
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.316	0.149	0.129	0.142	0.111	0.209	1.000

COPA, Top 1%

	C4	Math	ArXiv	Wiki	$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$
C4	1.000	0.357	0.311	0.307	0.299	0.328	0.259
Math	0.357	1.000	0.213	0.187	0.173	0.185	0.141
ArXiv	0.311	0.213	1.000	0.311	0.150	0.180	0.129
Wiki	0.307	0.187	0.311	1.000	0.193	0.209	0.137
$\nabla\mathcal{F}(\mathbf{w}_{\text{st.}})$	0.299	0.173	0.150	0.193	1.000	0.341	0.121
$\nabla\mathcal{F}(\mathbf{w}_{\text{mid}})$	0.328	0.185	0.180	0.209	0.341	1.000	0.235
$\nabla\mathcal{F}(\mathbf{w}_{\text{end}})$	0.259	0.141	0.129	0.137	0.121	0.235	1.000

Figure 14: Overlap ratio in top entries of gradient squares of **Mistral-7B** for C4, OpenWebMath, ArXiv, Wiki103, and $\nabla\mathcal{F}(\mathbf{w}_{\text{before FT}})$, $\nabla\mathcal{F}(\mathbf{w}_{\text{mid FT}})$, $\nabla\mathcal{F}(\mathbf{w}_{\text{after FT}})$ across 3 commonsense reasoning, 1 math, and 3 SuperGLUE tasks.

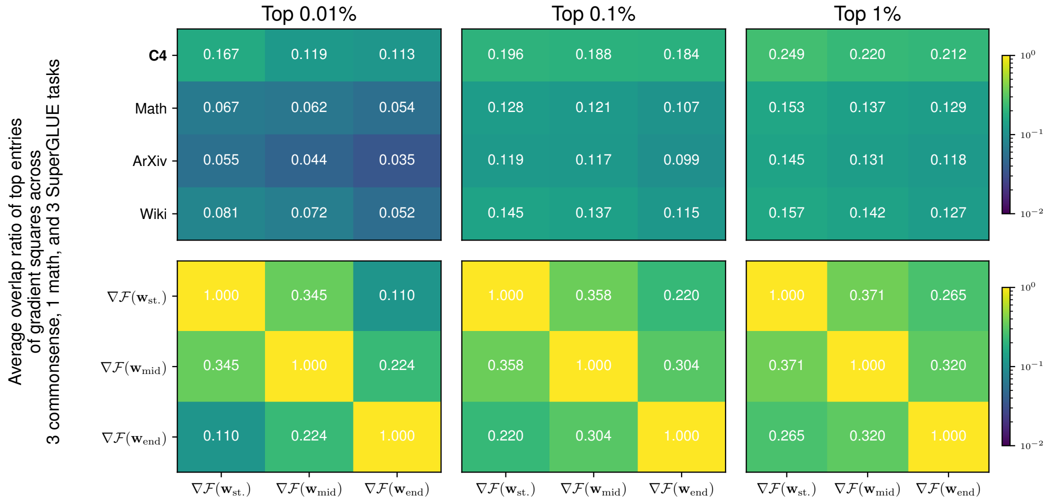


Figure 15: Average overlap ratio of top entries of gradient squares across 7 tasks in Figure 14.

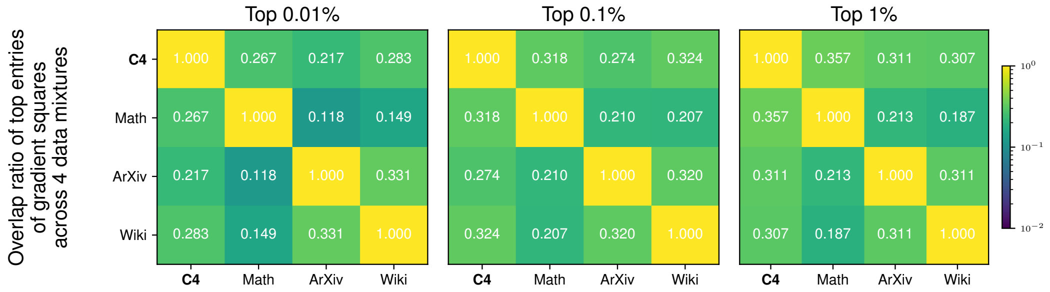


Figure 16: Overlap ratio of top entries of gradient squares among C4, OpenWebMath, ArXiv, and Wiki103.

G IMPLEMENTATION OF SPARSE OPERATIONS IN LINEAR LAYERS

Linear layers in LLMs often contribute most parameters (Kaplan et al., 2020). Since from Equation 7 we know

$$\mathbf{w}_{\text{sparse}} = \mathbf{w} \odot \mathbf{m}_k, \quad \mathbf{w}_{\text{dense}} = \mathbf{w} \odot (\mathbf{1}_d - \mathbf{m}_k), \quad \mathbf{w} = \mathbf{w}_{\text{sparse}} + \mathbf{w}_{\text{dense}} \tag{12}$$

and since $\mathbf{w}_{\text{dense}}$ would have the same shape (and the 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):

$$\begin{aligned}
\mathbf{w}\mathbf{x} &= \mathbf{w}_{\text{dense}}\mathbf{x} + \mathbf{w}_{\text{sparse}}\mathbf{x} && \qquad (13)\\
&= \text{SparseAddMM}(\text{DenseMM}(\mathbf{w}_{\text{dense}}, \mathbf{x}), \mathbf{w}_{\text{sparse}}, \mathbf{x}) \quad \text{faster with token generation} && \qquad (14)\\
&= (\mathbf{w}_{\text{dense}} + \mathbf{w}_{\text{sparse}})\mathbf{x} && \qquad (15)\\
&= \text{DenseMM}(\text{SparseAdd}(\mathbf{w}_{\text{sparse}}, \mathbf{w}_{\text{dense}}), \mathbf{x}) \quad \text{faster with ZO training} && \qquad (16)
\end{aligned}$$

```python
class SensitiveZOLinear(nn.Linear):
    w_sparse
    w_dense

    def forward_small_batched_decoding(self, X):
        # dense matmul
        dense_result = F.linear(X, self.w_dense, self.bias)
        # sparse addmm
        return torch.sparse.addmm(dense_result, X, self.w_sparse.T)

    def forward_large_batched_ZO_training(self, X):
        # sparse addition
        w = self.w_dense.add(self.w_sparse)
        # dense matmul
        return F.linear(X, w, self.bias)
```

Listing 1: Example PyTorch-like code snippet that implements the forward calls with 16-bit sparse and 16-bit dense parameters.

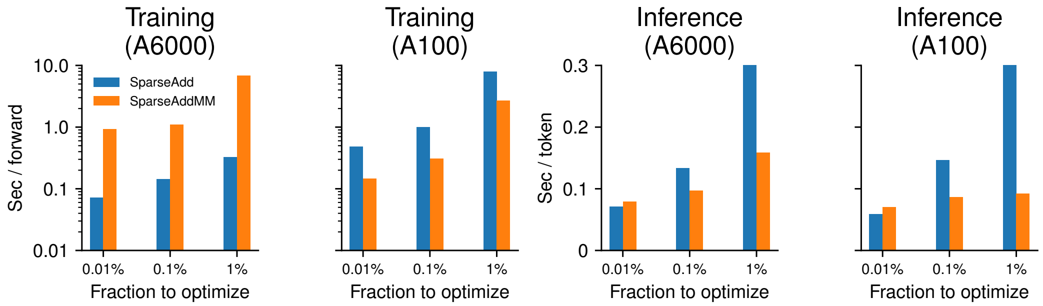


Figure 17: Time of SparseAdd (Equation 16) versus SparseAddMM (Equation 14) in Llama2-7B ZO training forward & inference. In subfigure 1 and 3, we use Nvidia RTX A6000 and Intel Xeno Gold 6342 CPUs, with PyTorch version 2.2, HuggingFace version 4.36, and CUDA 12.2. In subfigure 2 and 4, we use Nvidia 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 14 or Equation 16 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 14 would have a computational dependency over $\mathbf{x}$. During large-batched ZO training, $\mathbf{x}$ would be large enough such that Equation 14 would induce large computational overhead, as shown in subfigure 1 of Figure 17. In contrast, the computational complexity of Equation 16 is *independent* of $\mathbf{x}$ and when $\mathbf{x}$ is large, we would expect Equation 16 is *much* faster than Equation 14. 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 17.

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 17. 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 16 during large-batched ZO training and Equation 14 during small-batched autoregressive token generation.

In light of this observation, in our Figure 19, 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, 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 (Xi et al., 2023; Park et al., 2024). However, this creates difficulty on our "SparseAdd" approach as we will *violate the constraint of uniformly-spaced quantization bins* by computing $\mathrm{SparseAdd}(Q(\mathbf{w}_{\text{dense}}) + \mathbf{w}_{\text{sparse}})$. In this case, we also have 3 different implementations:

$$
\begin{aligned}
Q(\mathbf{w})\mathbf{x} &\sim Q(\mathbf{w}_{\text{dense}})\mathbf{x}+\mathbf{w}_{\text{sparse}}\mathbf{x} && (17)\\
&= \mathrm{SparseAddMM}\Big(\mathrm{Dequantize}\big(\mathrm{UniformMM}(Q(\mathbf{w}_{\text{dense}}),\mathbf{x})\big),\mathbf{w}_{\text{sparse}},\mathbf{x}\Big) \quad \text{uniform quantization} && (18)\\
&= \mathrm{SparseAddMM}\Big(\mathrm{Dequantize}(Q(\mathbf{w}_{\text{dense}})),\mathbf{x},\mathbf{w}_{\text{sparse}}\Big) \quad \text{similar to Equation 14} && (19)\\
&= (\mathrm{Dequantize}(Q(\mathbf{w}_{\text{dense}}))+\mathbf{w}_{\text{sparse}})\mathbf{x} && (20)\\
&= \mathrm{DenseMM}(\mathrm{SparseAdd}\,(\mathbf{w}_{\text{sparse}},\mathrm{Dequantize}(Q(\mathbf{w}_{\text{dense}})),\mathbf{x}) \quad \text{similar to Equation 16} && (21)
\end{aligned}
$$

Equation 18 would compute $\mathrm{UniformMM}(Q(\mathbf{w}_{\text{dense}}),\mathbf{x})$ *without* dequantizing $Q(\mathbf{w}_{\text{dense}})$ to 16 bits. This would make "SparseAdd" approach infeasible and we can only employ "SparseAddMM" approach in this case. Notice that both Equation 19 and Equation 21 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.

```python
class SensitiveZOLinear(nn.Linear):
    w_sparse
    w_dense_quantized

    def dequantize(self, w_quantized):
        ...
        return w_16_bit

    def forward_uniform_matmul(self, X):
        # fast uniform quantization matmul
        dense_result = uniform_quantized(X, self.w_dense_quantized.T)
        # sparse addmm
        return torch.sparse.addmm(dense_result, X, self.w_sparse.T)

    def forward_small_batched_decoding(self, X):
        # dequantize the dense weights
        w_dense_16_bit = self.dequantize(self.w_dense_quantized)
        # dense matmul
        dense_result = F.linear(X, w_dense_16_bit, self.bias)
        # sparse addmm
        return torch.sparse.addmm(dense_result, X, self.w_sparse.T)

    def forward_large_batched_ZO_training(self, X):
        # dequantize the dense weights
        w_dense_16_bit = self.dequantize(self.w_dense_quantized)
        # dense matmul
        dense_result = F.linear(X, w_dense_16_bit, self.bias)
        # sparse addmm
        return torch.sparse.addmm(dense_result, X, self.w_sparse.T)
```

Listing 2: Example PyTorch-like code snippet that implements the forward calls with 16-bit sparse and quantized dense parameters.

We recommend using Equation 18 when we use efficient uniform integer matmul to compute $Q(\mathbf{w}_{\textbf{dense}})\mathbf{x}$ and in other cases, using Equation 19 or Equation 21 follows our previous recommendation based on the size of input vectors.

H WALL-CLOCK TIME EFFICIENCY FROM SENSITIVE ZO'S STATIC SPARSITY

Static sparse ZO fine-tuning (Equation 7 and 8) can be written as a parameter-efficient perturbation & update rule:

$$\mathbf{w}_{\text{sparse}} = \mathbf{w} \odot \mathbf{m}_k,\ \ \mathbf{w}_{\text{dense}} = \mathbf{w} \odot (\mathbf{1}_d - \mathbf{m}_k) \tag{22}$$

$$\mathbf{w}_{\text{sparse},t+1} = \mathbf{w}_{\text{sparse},t} - \eta_t \hat{g}(\mathbf{w}_{\text{sparse},t}, (\mathbf{x}_t, y_t), \mathbf{z}_{k,t}) \tag{23}$$

Under the extreme sparsity regime (0.1% sparsity), this parameter-efficient rule will have wall-clock time efficiency in both ZO training and token generation process.

Comparison with ZO Full FT: faster in training. By employing parameter-efficient ZO fine-tuning with extreme sparsity, we also achieve 1.2 - 2.5× wall-clock time convergence speedup compared with ZO full fine-tuning as we nearly eliminate the ZO perturbation and optimizer update time, as Figure 18 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. 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.

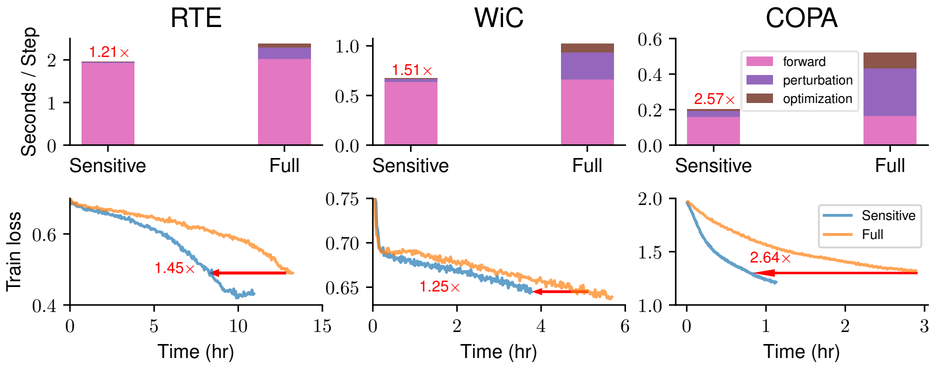


Figure 18: Iteration-wise time & wall-clock convergence time of sensitive ZO fine-tuning with static sparsity ("Sensitive") versus ZO full fine-tuning ("Full") for Llama2-7B. Here we use the 16-bit model as the base model for fine-tuning.

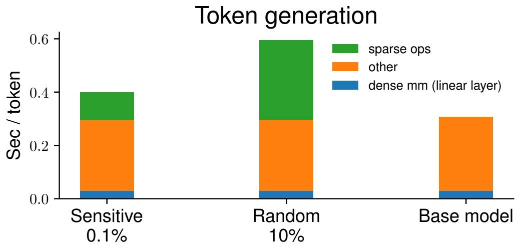


Figure 19: Inference speed of Llama2-7B with 0.1% sensitive parameters, 10% random subset sparse parameters, and the base model (no static sparsity extraction). The implementation of 0.1% sensitive parameters and 10% random subset sparse parameters are discussed in Appendix G.

Comparison with other sparsity methods: faster in inference. As the sensitive sparse fine-tuning method achieves great performance via optimizing only *0.1%* parameters (performance comparable to ZO full fine-tuning and 10% random subsets), during inference we achieve an end-to-end 1.49× speedup, with 2.15× speedup at sparse operations compared to 10% random subsets.

I SUPPLEMENTARY EXPERIMENT DETAILS

I.1 HYPERPARAMETERS IN EXPERIMENTS

For all ZO experiments, we use 20,000 training steps with ZO-SGD optimizer (Definition 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 7 and Table 8), we use batch size of 16. This experiment setting is identical to Malladi et al. (2023a).

Table 7: The chosen hyperparameters for experiments in Table 1. We repeat each hyperparameters for 3 random trials and report the average and standard deviation in Table 1.

(a) **Llama2-7B**

	Methods	SST-2	RTE	CB	BoolQ	WSC	WiC	COPA	WinoG	Wiki2
Q, ZO	**SensZOQ** (ϵ =1e-3)	5e-7	1e-6	1e-6	1e-6	5e-7	1e-6	1e-6	2e-6	5e-6
	LoRA (ϵ =1e-3)	1e-5	5e-5	1e-5	2e-5	1e-5	2e-5	1e-5	5e-5	1e-4
	Prefix (ϵ =1e-2)	1e-4	2e-4	5e-4	5e-4	1e-4	5e-4	2e-4	1e-3	1e-3
ZO	Full FT (ϵ =1e-3)	5e-7	5e-7	5e-7	5e-7	2e-7	5e-7	5e-7	1e-7	2e-6
	ICL (#examples)	16	16	16	8	16	8	8	16	NA
	(b) **Mistral-7B**									
Q, ZO	**SensZOQ** (ϵ =1e-4)	2e-8	5e-8	2e-8	2e-8	1e-8	2e-8	2e-8	1e-7	1e-7
	LoRA (ϵ =1e-4)	2e-6	5e-6	2e-6	2e-6	2e-6	2e-6	2e-6	1e-5	2e-5
	Prefix (ϵ =1e-3)	1e-3	2e-3	1e-3	1e-2	5e-4	1e-3	5e-4	2e-4	1e-2
ZO	Full FT (ϵ =1e-4)	2e-8	2e-8	1e-8	1e-8	1e-8	1e-8	2e-8	5e-8	1e-7
	ICL (#examples)	4	8	4	16	4	4	8	8	NA
	(c) **OPT-6.7B**									
Q, ZO	**SensZOQ** (ϵ =1e-3)	2e-7	5e-7	5e-7	5e-7	2e-7	5e-7	2e-7	1e-6	1e-6
	LoRA (ϵ =1e-3)	1e-5	2e-5	1e-5	2e-5	1e-5	2e-5	2e-5	5e-5	1e-4
	Prefix (ϵ =1e-2)	2e-3	1e-2	1e-3	5e-3	5e-3	1e-2	5e-3	2e-2	2e-1
ZO	Full FT (ϵ =1e-3)	2e-7	2e-7	2e-7	2e-7	2e-7	2e-7	5e-7	5e-7	1e-6
	ICL (#examples)	16	4	16	16	16	8	16	16	NA

Our hyperparameters (learning rate η, perturbation scaling constant ϵ, and the number of ICL examples) for Table 1 is reported in Table 7. For Table 1, we use constant η and ϵ throughout our experiments. We also report the chosen hyperparameter for Figure 5a and Figure 5b in Table 8. 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 except for Wiki2 that we always use 20 (as the best performance from $\{5, 20, 50, 100\}$).

For the smaller FO-Adam experiment in Table 2, we use the same codebase and we report the used learning rates in Table 9. We use a batch size of 8 and train for 1000 steps. We use the Adam optimizer with linear learning rate decay to 0 and no weight decay. We evaluate the model's performance at the end of 1000-step training.

Table 8: The chosen hyperparameters for experiments in Figure 5a and Figure 5b. We repeat each hyperparameters for 3 random trials and report the average to draw a line in Figure 5a and Figure 5b, 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
Weights with largest magnitude (static) (ϵ =1e-3)	2e-3	1e-3	2e-4	5e-5	1e-5
Weights with smallest magnitude (static) (ϵ =1e-3)	2e-3	5e-4	1e-4	1e-5	1e-6
smallest GraSP scores (static) (ϵ =1e-3)	2e-5	5e-6	2e-6	2e-6	1e-6

(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
Weights with largest magnitude (static) (ϵ =1e-3)	1e-3	5e-4	2e-4	1e-4	2e-5
Weights with smallest magnitude (static) (ϵ =1e-3)	1e-3	5e-4	1e-4	1e-5	2e-6
smallest GraSP scores (static) (ϵ =1e-3)	2e-5	1e-5	5e-6	2e-6	2e-6

(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
Weights with largest magnitude (static) (ϵ =1e-3)	1e-3	5e-4	5e-4	1e-4	1e-5
Weights with smallest magnitude (static) (ϵ =1e-3)	2e-3	5e-4	2e-5	2e-6	2e-6
smallest GraSP scores (static) (ϵ =1e-3)	5e-6	5e-6	1e-6	2e-6	1e-6

Table 9: The chosen hyperparameters for experiments in Table 2. We repeat each hyperparameters for 3 random trials and report the average and standard deviation in Table 2.

	Methods	SST-2	RTE	CB	BoolQ	WSC	WiC	COPA	WinoG	Wiki2
FO-Adam	Full FT	1e-5	2e-5	1e-5	1e-5	2e-5	1e-5	1e-5	2e-5	5e-5

I.2 TASK-SPECIFIC PROMPTS IN EXPERIMENTS

We describe our task templates in Table 10.

Table 10: 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>
WinoGrande (Llama2-7B, Mistral-7B, OPT-6.7B)	<context> <option>

I.3 ON-DEVICE MEMORY CONSTRAINTS

As illustrated in Table 11, a wide range of mobile or edge devices impose a memory constraint of **8 GiB**, which is the our target when we develop our SensZOQ in Section 3.2.

Table 11: 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., 2024a)	12 GiB
Samsung Galaxy S10 with Mali-G76 GPU (Gim & Ko, 2022)	8 GiB

I.4 HARDWARE, PLATFORM, LIBRARIES, AND OTHER DETAILS FOR FINE-TUNING AND BENCHMARKING

Figure 17 (subfigure 1 and 3), Figure 18, and Figure 19 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 17, 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 7 and Figure 11) we use the BrainFloat16 datatype (torch.bfloat16) to avoid the NaN issue from the Float16 datatype.

In Figure 17, 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 (same for Figure 19) 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:* ". We still use the Float16 datatype (torch.float16) for both benchmarks.