RoSA: Accurate Parameter-Efficient Fine-Tuning via Robust Adaptation

Mahdi Nikdan [*1] **Soroush Tabesh** [*1] **Elvir Crnčević** [1 2] **Dan Alistarh** [1 3]

Abstract

We investigate parameter-efficient fine-tuning (PEFT) methods that can provide good accuracy under limited computational and memory budgets in the context of large language models (LLMs). We present a new PEFT method called Robust Adaptation (RoSA) inspired by robust principal component analysis that jointly trains *low-rank* and *highly-sparse* components on top of a set of fixed pretrained weights to efficiently approximate the performance of a full-fine-tuning (FFT) solution. Across a series of challenging generative tasks such as grade-school math and SQL query generation, which require fine-tuning for good performance, we show that RoSA outperforms LoRA, pure sparse fine-tuning, and alternative hybrid methods at the same parameter budget, and can even recover the performance of FFT on some tasks. We provide system support for RoSA to complement the training algorithm, specifically in the form of sparse GPU kernels which enable memory- and computationally-efficient training, and show that it is also compatible with low-precision base weights, resulting in the first joint representation combining quantization, low-rank and sparse approximations. Our code is available at `https://github.com/IST-DASLab/RoSA`.

1. Introduction

The advances brought about by large language models (LLMs) come with very large computational and memory costs, especially for training such models from scratch. In this context, fine-tuning LLMs using limited data has become an effective and popular approach to improve performance on specific tasks, e.g. (Wei et al., 2021; Ouyang et al., 2022; Wang et al., 2022a; Liu et al., 2022), or adapt LLMs to better fit expected user behavior (Askell et al., 2021; Bai et al., 2022). Yet, full fine-tuning of all LLM parameters (FFT), can be extremely expensive, especially in terms of memory cost, rendering this process prohibitive.

Parameter-Efficient Fine-Tuning (PEFT) methods address this issue by allowing users to optimize only over a restricted set of parameters, relative to the original model. On the one hand, this allows partial accuracy recovery relative to FFT, at a fraction of its computational and memory cost. An extremely popular recent instance of PEFT in the context of LLMs is given by the Low-Rank Adaptation (LoRA) family of methods (Hu et al., 2021), which train low-rank "adapter" layers for a selection of the model layers. LoRA methods are based on the intuition that the fine-tuning updates of pre-trained LLMs have low "intrinsic rank" during specialization to a sub-task, which allow these updates to be well-approximated by adapters. Besides memory and computational cost reductions, low-rank adaptation also has the advantage of implicit regularization, which can lead to more stable training, and simplify hyper-parameter search.

One key weakness of LoRA-type methods is the fact that they can fail to recover accuracy for "harder" fine-tuning tasks, relative to FFT. This accuracy gap, illustrated in Figure 2, appears more likely to occur when the target tasks is more complex, such as the case for mathematical reasoning or coding tasks. It is therefore still an open question whether there exist PEFT methods which combine the good practical performance and ease-of-use of LoRA-type methods with the high accuracy of FFT.

Contribution. In this paper, we take a step towards addressing this question, by proposing a new PEFT method called **Ro**bu**S**t **A**daptation (**RoSA**). RoSA has similar computational and memory cost relative to LoRA-type methods, but is significantly more accurate at similar parameter and computational budgets, while being easy to use and tune. Specifically, in practical experiments RoSA essentially matches the accuracy of full fine-tuning, while offering stable convergence and relatively simple hyper-parameter tuning. We complement these algorithmic observations with a practical implementation, showing that RoSA preserves the memory

[*]Equal contribution [1]ISTAustria [2]Graz University of Technology [3]Neural Magic. Correspondence to: Mahdi Nikdan <mahdi.nikdan@ista.ac.at>, Soroush Tabesh <soroush.tabesh@ista.ac.at>, Dan Alistarh <dan.alistarh@ista.ac.at>, Elvir Crnčević <elvir.crncevic@ista.ac.at>.

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

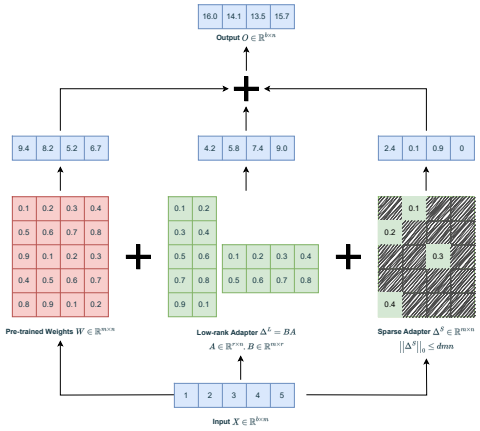


Figure 1: Illustration of Robust Adaptation (RoSA) applied to a single FC layer: In this instance, the weight matrix is of dimensions 5×4 and the batch size is 1. The low-rank adapter has a rank of 2, and the sparse adapter has a density of 20%. Trainable parameters are depicted in green, while red indicates parameters that remain frozen.

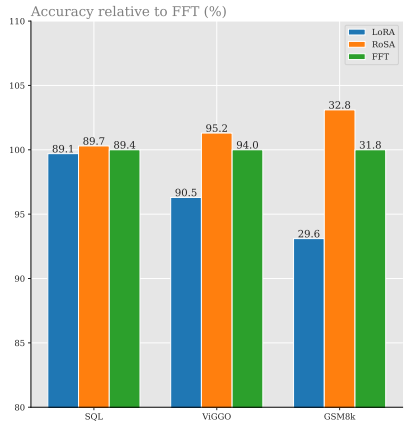


Figure 2: Comparison of the highest achieved accuracy by a single-epoch adaptation using various methods across three datasets on LLaMA2-7B, taken from our main experiments in Table 1. (While LoRA and RoSA store parameters in `bfloat16` (Dean et al., 2012) we use `float32` for FFT since they are more stable). Each bar shows the percentage of accuracy relative to the accuracy achieved by FFT, and the numbers on top of the bars indicate the absolute accuracy.

advantage of LoRA-type methods.

The motivation behind RoSA comes by revisiting the low "intrinsic rank" assumption that is the basis for the LoRA family of methods. Specifically, our investigation across several tasks shows that, while the FFT update can indeed be well approximated by a low-rank matrix, one can obtain a significantly better fit via a *low-rank plus sparse matrix*, especially in the case of more complex tasks. Intuitively, the latter representation is better suited to matching outlier components which can cause a significant fraction of the compression error in the context of LLMs (Dettmers et al., 2022; 2023b). This observation provides a connection to the area of robust principal component analysis (robust PCA) (Candès et al., 2011), which postulates that matrices arising from a noisy series of measurements can often be approximated as a sum between a low-rank component and a sparse one, and investigates algorithms for recovering such matrices. Starting from the hypothesis that the sum of gradient updates corresponding to FFT can be seen as an instance of robust PCA, we investigate methods for recovering such a sparse plus low-rank representation during training.

Concretely, our proposed scheme trains *two adapters*: a standard low-rank adapter, complemented by a sparse adapter, which are trained "in parallel" relative to the original pre-trained weights. The challenge is threefold, since we have to: 1) identify a highly-performant sparsity mask; 2) find a co-training mechanism which yields stable convergence; and, 3) provide system support, specifically for an efficient sparse backward pass.

Building on prior work in the area (Sung et al., 2021; Chen et al., 2021), we resolve all three challenges and show that RoSA adapters can lead to considerably higher accuracy of the resulting model, at a comparable parameter, memory, and computational budget relative to standard adapters that are either low-rank or sparse. We complement our algorithmic contribution with an efficient system implementation of RoSA in Pytorch, that is fast on NVIDIA GPUs. Specifically, supporting sparse adapters with low memory and computational overhead is non-trivial, as we must leverage sparse representations that are notoriously hard to support efficiently on GPUs (Gale et al., 2020).

In addition, we extend our approach to support quantization of the base weights via QLoRA (Dettmers et al., 2023a), further improving efficiency at little or no accuracy cost. This results in a joint representation which recovers accuracy by combining all three common forms of compression: quantization, low-rank projections, and sparsity.

In summary, we present promising evidence that the accuracy gap between adaptation methods and full fine-tuning of LLMs can be significantly reduced or even eliminated in some cases, without sacrificing practical accessibility. Therefore, RoSA can be an additional technique in the toolbox of machine learning practitioners working with LLMs in resource-constrained settings.

2. Related Work

Parameter-Efficient Fine-Tuning. Recent open LLMs (Touvron et al., 2023a;b; Zhang et al., 2022;

MosaicML, 2023b) have demonstrated strong performance across various NLP tasks, but present challenges during training and inference due to high memory and computation cost. The common practice is to fine-tune these models on smaller downstream tasks rather than training from scratch (Min et al., 2021; Wei et al., 2021; Ouyang et al., 2022; Wang et al., 2022b;a; Liu et al., 2022). While this approach partially addresses the computation demands, memory requirements are still a major concern. Parameter-Efficient Fine-Tuning (PEFT) methods have emerged as a solution (Hu et al., 2021; Zhang et al., 2023; Li & Liang, 2021; Liu et al., 2021; 2023; Lester et al., 2021; Liu et al., 2022; Sanh et al., 2021; Hyeon-Woo et al., 2021; Edalati et al., 2022; Li et al., 2023; Qiu et al., 2023; Sung et al., 2021): Instead of fine-tuning all parameters, they selectively fine-tune smaller sets of parameters, potentially including a subset of the original ones. Notably, LoRA-type methods (Hu et al., 2021; Zhang et al., 2023), which train a low-rank perturbation to the original weights, have gained popularity for their efficiency and ease of use (Dettmers et al., 2023a). However, it is known that they often fail to recover the accuracy of FFT (Edalati et al., 2022; Zhang et al., 2023).

Earlier work focused on smaller-scale BERT-type models and sparse and/or low-rank updates. Specifically, FISH Mask (Sung et al., 2021) updates only a sparse subset of weights in the BERT-base model (Devlin et al., 2018). Its reliance on the Fisher Information Matrix (FIM) for generating sparsity masks renders it impractical for LLMs, unless heavy approximations are employed. FISH Mask uses the empirical diagonal estimation of the FIM. We examine its validity in Section 5, and find it to be less effective in the case of LLMs. Relatedly, DSEE (Chen et al., 2021) trains a combination of low-rank and sparse adapters. However, despite promising results on BERT models, we find DSEE faces two main challenges in our setting. First, the DSEE sparsity masks perform a *task-independent* decomposition of pre-trained weights. As we demonstrate in Section 5, this mask generation method does not effectively outperform random masks in the context of LLMs, and significantly underperforms RoSA masks, even when applied to gradients instead of weights. Second, DSEE lacks system support for reducing costs by using a sparse adapter. In contrast, RoSA comes with efficient GPU support, and is also compatible with weight quantization, as we show in QRoSA.

Sparse Training / Fine-Tuning. Sparsity in language models has emerged as a popular strategy to address their significant computational and memory demands (Hoefler et al., 2021), both for inference (Gale et al., 2019; Singh & Alistarh, 2020; Sanh et al., 2020; Frantar & Alistarh, 2022) and training (Evci et al., 2020; Peste et al., 2021; Hubara et al., 2021; Jiang et al., 2022; Nikdan et al., 2023). A related research direction is sparse fine-tuning, where a network, pre-trained and sparsified on an upstream dataset, undergoes fine-tuning on a downstream task while keeping the sparsity mask fixed (Nikdan et al., 2023; Kurtic et al., 2022; 2023). Despite both sparse fine-tuning and sparse adaptation optimizing over a fixed subset of parameters, in sparse fine-tuning, the weights not involved are pruned (set to zero), whereas in sparse adaptation, they are merely frozen. This distinction allows us to achieve extremely high sparsity levels in sparse adaptation masks (over 99%, see Section 5), whereas sparse training / fine-tuning typically struggles to 90-95% without significant accuracy loss.

Robust Principal Component Analysis (RPCA). RPCA is a well-explored domain, focusing on techniques that can effectively handle data corrupted by outliers or gross errors. While classical Principal Component Analysis (PCA) assumes that the data is clean, RPCA methods extract robust principal components even in the presence of significant outliers (Gnanadesikan & Kettenring, 1972; Fischler & Bolles, 1981; Wright et al., 2009; Candès et al., 2011; De La Torre & Black, 2003; Huber, 2004; Ke & Kanade, 2005). Specifically, given noisy measurements expressed as $A = L + S$, where L is low-rank and S is sparsely supported with elements of arbitrary large magnitude, the goal is to recover L and S. While early approaches did not achieve this in polynomial time (De La Torre & Black, 2003; Huber, 2004; Ke & Kanade, 2005; Gnanadesikan & Kettenring, 1972; Fischler & Bolles, 1981), recent papers show that it is possible to relax this by substituting the low-rank constraint on L with a constraint on its nuclear norm (Wright et al., 2009; Candès et al., 2011). By contrast, we perform Robust PCA-type optimization over a series of adapter matrices that are being learned jointly in an LLM. As such, existing theoretical mechanisms do not apply, although extending them would be an interesting question for future work.

System Support for Sparsity. While PyTorch (Paszke et al., 2019) and STen (Ivanov et al., 2022) have recently incorporated partial sparsity support for inference, obtaining benefits from unstructured sparse representations–as needed in our work–is notoriously challenging, especially on GPU hardware. So far, Sputnik (Gale et al., 2020) is the only library to provide speedups in this context, although structured representations are known to be more amenable to speedups (Gray et al., 2017; Castro et al., 2023; Li et al., 2022). In this context, our kernels provide significant improvements upon Sputnik in the unstructured sparsity case by using a better indexing scheme and introducing a sparsity-adaptive SDDMM kernel for the backward pass.

3. Adaptation of Large Language Models

3.1. Notation

Let $\mathcal{N}$ represent a pre-trained Large Language Model (LLM), and let $\mathcal{W} = \{\boldsymbol{W}_1, \boldsymbol{W}_2, ..., \boldsymbol{W}_k\}$ denote a se-

quence of layers containing all fully connected weights of $\mathcal{N}$, including sub-attention layers, with $\boldsymbol{W}_i \in \mathbb{R}^{m_i \times n_i}$ for all $1 \leq i \leq k$. Let the vector $\bar{w} \in \mathbb{R}^{\bar{d}}$ indicate the rest of $\mathcal{N}$'s parameters (biases, normalization parameters, etc.) concatenated into a single vector. Given a dataset $\mathcal{D}$ and a loss function $\mathcal{L}(\mathcal{D}; \mathcal{W}, \bar{w})$, full fine-tuning (FFT) of $\mathcal{N}$ on $\mathcal{D}$ can be formulated as solving the optimization problem:

$$\min_{\mathcal{W}, \bar{w}} \mathcal{L}(\mathcal{D}; \mathcal{W}, \bar{w}) \tag{1}$$

Given that LLMs typically contain billions of parameters, performing FFT can be slow and computationally expensive. This often renders it challenging or even impossible to execute on standard GPUs. A solution to this involves the application of adapters, which we will now formulate. Let $\Delta = \{\boldsymbol{\Delta}_1, \boldsymbol{\Delta}_2, ..., \boldsymbol{\Delta}_k\}$ include perturbations to the original fully connected weights, where $\boldsymbol{\Delta}_i \in \mathbb{R}^{m_i \times n_i}$ for all $1 \leq i \leq k$. Define $\mathcal{W} + \Delta = \{\boldsymbol{W}_1 + \boldsymbol{\Delta}_1, \boldsymbol{W}_2 + \boldsymbol{\Delta}_2, ..., \boldsymbol{W}_k + \boldsymbol{\Delta}_k\}$. Additionally, let vector $\bar{\delta} \in \mathbb{R}^{\bar{d}}$ denote a perturbation to $\bar{w}$. The *adapted* parameters are then found by solving the following optimization problem:

$$\min_{\Delta, \bar{\delta}} \mathcal{L}(\mathcal{D}; \mathcal{W} + \Delta, \bar{w} + \bar{\delta}), \quad s.t. \quad \mathcal{C}(\Delta, \bar{\delta}) \tag{2}$$

where $\mathcal{C}(\Delta, \bar{\delta})$ is a set of constraints on the perturbations, such as low-rank or sparse, aiming to reduce the memory requirements or computational complexity of the optimization problem. Note that an adaptation with no constraints is equivalent to FFT.

In this context, our exclusive focus is on adaptations where $\bar{\delta} = \mathbf{0}$, as it aligns with standard practice. Nevertheless, given that $\bar{w}$ typically contains significantly fewer parameters than $\mathcal{W}$, there is room for fine-tuning $\bar{w}$ as well. Also, we are specifically focusing on cases where all fully connected weights undergo adaptation, but our arguments extend trivially to the case where only a subset of these weights is being adapted. We now discuss a few special cases.

LoRA: Low-Rank Adaptation. The well-known Low-Rank Adaptation (LoRA) (Hu et al., 2021) constrains the perturbations in Δ to exhibit a low rank, specifically the optimization objective will be the following:

$$\begin{aligned} &\min_{\Delta} \mathcal{L}(\mathcal{D}; \mathcal{W} + \Delta, \bar{w}), \\ &s.t. \quad \forall\, 1 \leq i \leq k : \mathit{rank}(\boldsymbol{\Delta}_i) \leq r \end{aligned} \tag{3}$$

with r being a fixed small number. This approach reduces the number of trainable weights for layer i from $m_i n_i$ to $r(m_i + n_i)$, resulting in more memory-efficient fine-tuning.

SpA: Sparse Adaptation. Sparse Adaptation (SpA), e.g. (Sung et al., 2021), imposes high sparsity constraints on perturbations, i.e., the optimization objective will be:

$$\begin{aligned} &\min_{\Delta} \mathcal{L}(\mathcal{D}; \mathcal{W} + \Delta, \bar{w}), \\ &s.t. \quad \forall\, 1 \leq i \leq k : ||\boldsymbol{\Delta}_i||_0 \leq d m_i n_i \end{aligned} \tag{4}$$

Algorithm 1 Mask Generation

Require: $\mathcal{W}, \bar{w} \leftarrow$ the fully connected weights and the rest of the LLM parameters, respectively
Require: $\mathcal{D}_{\mathcal{M}} \leftarrow$ the mask generation dataset, typically a small subset of the actual dataset
Require: $\mathcal{L}(.) \leftarrow$ the loss function
Require: $d \leftarrow$ mask density
Require: $\alpha \leftarrow$ gradient accumulation exponent
  $\mathcal{G} \leftarrow \{\mathbf{0}, \mathbf{0}, ..., \mathbf{0}\}$
  `[iterate through samples of` $\mathcal{D}_{\mathcal{M}}$`]`
  for $s \in \mathcal{D}_{\mathcal{M}}$ **do**
    `[calculate the gradients for this sample]`
    $\mathcal{G}^s, \bar{g}^s \leftarrow \nabla\mathcal{L}(s; \mathcal{W}, \bar{w})$
    `[accumulate the gradients]`
    $\mathcal{G} \leftarrow \mathcal{G} + (\mathcal{G}^s)^\alpha$
  end for
  for $G_i \in \mathcal{G}$ **do**
    `[top-k elements of the accumulated grads]`
    $\mathcal{M}_i \leftarrow$ *TopK-Mask*$(\mathcal{G}_i, d \times$ *numel*$(\mathcal{G}_i))$
  end for
  return $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_k\}$

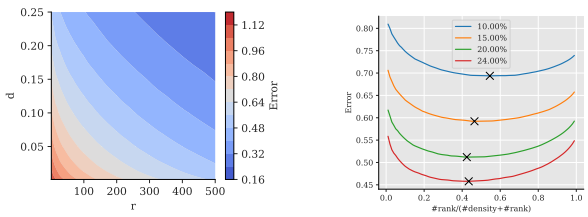
where $d < 1$ represents the perturbation density and $||.||_0$ denotes the ℓ_0 norm. It is common (Sung et al. (2021); Chen et al. (2021)) to consider the case where each perturbation has a fixed support throughout training. This way, SpA reduces the number of trainable parameters by a factor of d. At the same time, as discussed in Section 2, it encounters the primary challenges of 1) finding a good sparse support and 2) leveraging unstructured sparsity for speed and memory gains. Next, we discuss how our method approaches both challenges.

3.2. RoSA: Robust Adaptation

We now describe our main adaptation method.

Motivation. One key drawback of existing LoRA-type methods is that, when faced with more complex downstream tasks, they often fail to match full fine-tuning accuracy (see Figure 2.) Intuitively, this occurs because the low-rank prior may not be able to capture the structure of more complex updates in this case, filtering important directions. This filtering issue becomes particularly evident when conducting Singular Value Decomposition (SVD) on the FFT updates $\boldsymbol{\Delta}^*$ (defined as $\boldsymbol{\Delta}^* = \boldsymbol{W}^{\text{FFT}} - \boldsymbol{W}^{\text{BASE}}$) of LLM layers, as detailed in the Appendix D. These analyses reveal that while $\boldsymbol{\Delta}^*$ is rank-deficient (see Figure 7), it is not strictly low-rank. This distinction is characterized by the presence of a substantial fraction of singular values with relatively small, yet non-zero, magnitudes.

Robust Principal Component Analysis (RPCA) suggests an alternative in extracting *robust* principal components via a low-rank matrix L and a sparse matrix S. This decomposition offers a more nuanced approximation of the fine-tuning updates compared to solely low-rank methods.



(a) (b)

Figure 3: Illustration of the Frobenius norm error (Figure 3a) of a Robust PCA approximation to the full-fine-tuning update, for an arbitrary layer (`l:20, v_proj` of LLaMA2-7B, while varying rank and sparsity independently. Figure 3b depicts slices of Figure 3a with similar parameter counts, showcasing the trade-off between sparsity and low-rank under different parameter budgets.

To demonstrate the potential of using a combination of sparse and low-rank matrices to approximate a $\mathbf{\Delta}^*$ fine-tuning perturbation in the context of LLMs, we apply an RPCA solver to extract robust principal components $\tilde{\mathbf{\Delta}}^* = \tilde{\mathbf{\Delta}}^S + \tilde{\mathbf{\Delta}}^L$ of a randomly selected layer of LLaMA2-7B for a given sparsity and rank. In Figure 3a, we have analyzed a randomly selected module from LLaMA2-7B, computed its $\mathbf{\Delta}^*$ when fine-tuned on the GSM8k dataset, and then applied GreBsmo RPCA solver (Zhou & Tao, 2013), with varying ranks and densities for the low-rank and sparse components. The results in Figure 3b clearly demonstrate that, given a parameter budget to approximate $\mathbf{\Delta}^*$, *employing a combination of low-rank and sparse approximations yields a more accurate representation than using either approach in isolation*.

This analysis motivates our joint use of low-rank and sparse fine-tuning. The link between RPCA and RoSA lies in the former's introduction of the low-rank and sparse decomposition, a concept we leverage in RoSA to enhance the efficiency and accuracy of fine-tuning LLMs. In practice, our approach will do this in a task-adaptive fashion by "warming up" a LoRA instance for a short training interval and then identifying the largest sparse directions for improvement.

Formulation. We formulate the optimization objective of Robust Adaptation (RoSA) as follows:

$$\min_{\Delta^L, \Delta^S} \mathcal{L}(\mathcal{D}; \mathcal{W} + \Delta^L + \Delta^S, \bar{w}),$$
$$\text{s.t.} \quad \forall\, 1 \le i \le k : \begin{cases} rank(\mathbf{\Delta}_i^L) \le r \\ ||\mathbf{\Delta}_i^S||_0 \le dm_i n_i \end{cases} \tag{5}$$

where Δ^L and Δ^S represent the low-rank and sparse adapters, respectively. In practice, we generate the sparsity masks using Algorithm 1, and then optimize the low-rank and sparse adapters jointly. Refer to Figure 1 and Appendix Algorithm 2 for a detailed description of RoSA.

4. System Implementation

In this section, we briefly describe our efficient implementation of RoSA, detailed in full in Appendix A.

Low-Rank Format. Similar to Hu et al. (2021), we store an $m \times n$ low-rank adapter with rank r as the multiplication of two matrices $\boldsymbol{BA}$, where $\boldsymbol{B}$ and $\boldsymbol{A}$ are $m \times r$ and $r \times n$, respectively.

Sparse Format. Sparse adapters are stored in Compressed Sparse Row (CSR) format, which utilizes three lists to represent an $m \times n$ sparse matrix with nnz non-zero values: a `values` list with size nnz, storing the non-zero values; a `row-offsets` list with size $m + 1$, indicating the position of the first non-zero element in each row within the `values` list; and a `column-indices` list with size nnz, containing the column index of each corresponding element in the `values` list. Additionally, in line with Sputnik (Gale et al., 2020), an extra `row-indices` list with size m is included, sorting rows based on their non-zero element count. In our case, this `row-indices` list is employed for load-balancing and kernel launch configuration purposes.

Forward Pass. Consider a single fully connected layer with an adapted weight matrix $\boldsymbol{W} + \mathbf{\Delta}^L + \mathbf{\Delta}^S$ of size $m \times n$. For simplicity, assume there is no bias vector. Given a batch of inputs $\boldsymbol{X}$ of size $b \times m$, the layer output is expressed as:

$$\begin{aligned} \boldsymbol{O} &= \boldsymbol{X}(\boldsymbol{W} + \mathbf{\Delta}^L + \mathbf{\Delta}^S) \\ &= \boldsymbol{X}(\boldsymbol{W} + \mathbf{\Delta}^S) + (\boldsymbol{X}\boldsymbol{B}^L)\boldsymbol{A}^L \end{aligned} \tag{6}$$

Calculating the term $\boldsymbol{W} + \mathbf{\Delta}^S$ requires the addition of sparse and dense matrices, for which we provide an efficient kernel detailed in Appendix A. It is worth noting that the multiplication in the second term is decomposed into two multiplications with low-rank, making it extremely fast.

Backward Pass. Given the gradients of the output $\frac{\partial \mathcal{L}}{\partial \boldsymbol{O}}$, the backward pass through a layer involves calculating the gradients of the parameters and inputs, as follows:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \boldsymbol{X}} &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{O}}(\boldsymbol{W} + \mathbf{\Delta}^L + \mathbf{\Delta}^S)^T \\ &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{O}}(\boldsymbol{W} + \mathbf{\Delta}^S)^T + \Big(\frac{\partial \mathcal{L}}{\partial \boldsymbol{O}}(\boldsymbol{A}^L)^T\Big)(\boldsymbol{B}^L)^T \end{aligned} \tag{7}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{B}^L} = \frac{\partial \mathcal{L}}{\partial (\boldsymbol{B}^L \boldsymbol{A}^L)}(A^L)^T = X^T\Big(\frac{\partial \mathcal{L}}{\partial \boldsymbol{O}}(A^L)^T\Big) \tag{8}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{A}^L} = (B^L)^T \frac{\partial \mathcal{L}}{\partial (\boldsymbol{B}^L \boldsymbol{A}^L)} = \Big((B^L)^T X^T\Big)\frac{\partial \mathcal{L}}{\partial \boldsymbol{O}} \tag{9}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{\Delta}^S} = X^T \frac{\partial \mathcal{L}}{\partial \boldsymbol{O}} \tag{10}$$

Similarly to formula 6, Equations 7, 8, and 9 can also be computed efficiently. However, the implementation of equation 10 has a specific structure called a Sampled Dense-Dense Matrix Multiplication (`SDDMM`) (Nikdan et al., 2023), i.e. multiplying two dense matrices where only specific elements of the output are needed.

Leveraging Mask Structure. While general `SDDMM` is efficiently supported in e.g., `sputnik`, one special feature of our setting is that non-zero values in RoSA masks tend to cluster in a small subset of rows/columns, as illustrated in Appendix A. We suspect that this is correlated to the low-rank structure of the complementary adapter. To exploit this, we provide a new specialized `SDDMM` implementation which leverages this observation to maximize efficiency, specifically by dynamically skipping fully-zero rows and columns when present, depending on the specific sub-matrix structure. Compared to the SOTA `sputnik` kernels, our RoSA kernel achieves a geometric mean speedup of 1.36x and a peak speedup of 3x on LLM matrices. We provide a full discussion of matrix structure, kernel descriptions and layer-wise speedups in Appendix A.

Gradient Accumulation. As explained in Algorithm 1, creating the masks involves accumulating full gradients, which can be challenging in terms of memory. To address this, we adopt a simple solution by transferring the gradients of each weight matrix to CPU as soon as it is computed. This ensures that, at most, one weight matrix's gradient is stored on GPU at any given time. We note that this approach does not affect the runtime significantly, as the mask generation dataset is typically very small (32 samples in our experiments).

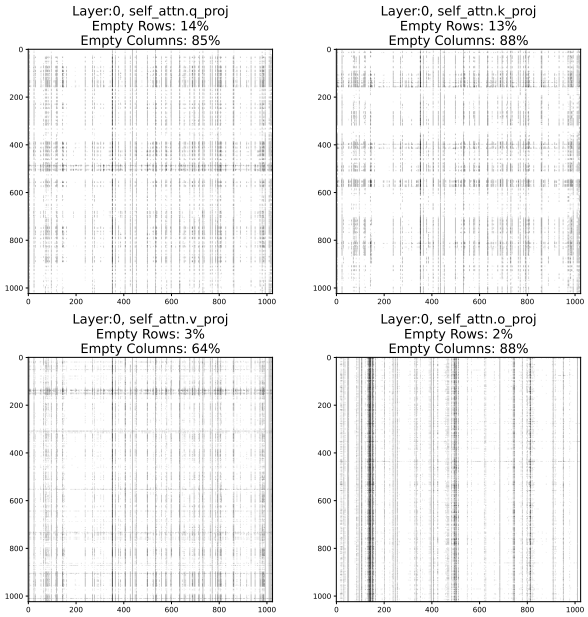


Figure 4: Illustration of row and column sparsity structure for the RoSA masks. Specifically, a subset of masks in the LLaMA2-7B model is visualized with a max-pool kernel of size 4 and stride 4, showing that a fraction of around 50% of the parameter rows and columns are completely zero.

5. Experiments

We now provide experimental support for the effectiveness of RoSA, and of QRoSA, its variant with quantized base weights. The following subsection outlines the experiment settings, including details on the network and datasets. To ensure a fair comparison, we conducted thorough and careful tuning for each adaptation method, details of which are described next. We then present the results, along with ablation studies, showcasing the improvements achieved by RoSA. Finally, we also assess RoSA's memory utilization, highlighting that it requires the same resources as LoRA and SpA in a fixed parameter budget while offering significantly improved accuracy.

5.1. Settings

Setup, Model and Datasets. We integrated RoSA into a fork of the standard `PEFT` library (Mangrulkar et al., 2022) and performed all the experiments using the MosaicML `llm-foundry` codebase (MosaicML, 2023a). We perform fine-tuning of the LLaMA2-7B model (Touvron et al., 2023b) on three standard datasets: ViGGO (Juraska et al., 2019), GSM8k (Cobbe et al., 2021), and SQL generation (Zhong et al., 2017; Yu et al., 2018), containing $5.1k$, $7.47k$, and $30k$ training samples and $1.08k$, $1.32k$, and $1k$ test samples, respectively. Refer to Appendix F for examples of the GSM8k dataset. In the case of SQL, we follow the dataset formation strategy described in (Niederfahrenhorst et al., 2023). On GSM8k, we only consider the accuracy of the final answer. Notably, these datasets are chosen such that they are highly specialized and, therefore, require fine-tuning for good performance: for example, on GSM8k, the pre-trained LLaMA-2 model has 0% one-shot accuracy, and the multi-shot accuracy is also very poor (around 6%).

Hyperparameters. In all experiments, we use a standard batch size of 32 (micro-batch size 1 + gradient accumulation) and a maximum context length of 512, which matches the dataset sample structure. We employ the AdamW optimizer (Loshchilov & Hutter, 2017) with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, and a linear learning rate scheduler with 20 batches warmup. Notably, all floating-point values are stored in `bfloat16` (Dean et al., 2012), popular due to low memory usage and good accuracy. Our main experiments run for a single epoch, but we demonstrate in ablation studies that extended training can further improve adaptation results. Following (Hu et al., 2021), we

Table 1: Comparison of fine-tuning LLaMA2-7B using FFT, LoRA, SpA, and RoSA in terms of memory usage and accuracy on three datasets. For RoSA, we examine different splits of the parameter budget into low-rank and sparse adapters. (†) Our experiments show that the single-epoch FFT results on ViGGO are suboptimal when the parameters are stored in `bfloat16`. Single-epoch `float32` FFT results on GSM8k, ViGGO, and SQL are 31.8, 94.0, and 89.4, respectively.

			GSM8k		ViGGO		SQL
	#Params	Memory	1 Epoch	Extended	1 Epoch	Extended	1 Epoch
FFT	6.7 B	> 60 GB	**32.3**	**38.8**	**82.1**	**95.0**	**89.0**
LoRA $r = 16$	41.1 M	20.6 GB	28.4	**37.8**	90.5	95.8	88.7
RoSA $r = 12, d = 0.15\%$	41.0 M	20.3 GB	**31.2**	36.0	**95.0**	96.5	88.3
RoSA $r = 8, d = 0.3\%$	40.8 M	20.3 GB	29.2	37.5	94.5	**97.1**	77.6
RoSA $r = 4, d = 0.45\%$	40.6 M	20.3 GB	30.6	35.5	93.4	96.6	**89.7**
SpA $d = 0.6\%$	40.4 M	20.3 GB	26.2	29.5	72.6	89.8	83.2
LoRA $r = 32$	82.3 M	20.9 GB	29.6	36.2	87.0	96.8	**89.1**
RoSA $r = 24, d = 0.3\%$	81.9 M	20.6 GB	30.5	37.8	94.4	95.8	88.9
RoSA $r = 16, d = 0.6\%$	81.6 M	20.7 GB	**32.2**	**38.6**	**95.2**	**97.1**	88.3
RoSA $r = 8, d = 0.9\%$	81.2 M	20.7 GB	30.3	37.2	94.5	96.9	88.9
SpA $d = 1.2\%$	80.9 M	20.7 GB	21.9	29.9	45.8	95.7	74.2
LoRA $r = 64$	164.5 M	21.7 GB	27.4	35.5	76.9	95.0	88.7
RoSA $r = 48, d = 0.6\%$	163.8 M	21.3 GB	30.5	38.2	93.0	96.6	88.1
RoSA $r = 32, d = 1.2\%$	163.1 M	21.4 GB	32.2	36.2	93.4	**97.3**	**89.2**
RoSA $r = 16, d = 1.8\%$	162.4 M	21.5 GB	**32.8**	**38.4**	**95.1**	96.5	84.6
SpA $d = 2.4\%$	161.7 M	21.8 GB	29.6	37.2	92.3	95.7	87.8

use $\alpha = 16$ and a dropout of 0.05 for the low-rank adapter, while experimenting with various r values ranging from 4 to 64. Additionally, we set the size of the mask generation dataset to 32 samples in all experiments while tuning the gradient accumulation exponent (α in Algorithm 1) as a binary hyperparameter (1 for averaging gradients and 2 for diagonal Fisher).

The sparse adapter's density ranges from 0.15% to 2.4%. While it is possible to adapt only a subset of the linear layers in the model, we specifically consider the case where every fully connected layer undergoes adaptation. This choice is motivated by the significantly lower memory usage of adaptation parameters compared to storing the original parameters (see Tables 1 and 2). The best learning rates for single-epoch FFT are 4×10^{-5}, 2×10^{-5}, and 1×10^{-4} on SQL, ViGGO, and GSM8k, respectively, while for extended FFT it is 4×10^{-5} on ViGGO and 5×10^{-5} on GSM8k. For LoRA and SpA parameters, the best-performing learning rates are selected in the range $[10^{-4}, 10^{-3}]$ and $[10^{-4}, 8 \times 10^{-4}]$, respectively. In RoSA experiments, we find it beneficial to initially fine-tune solely with LoRA for 64 batches, generate and fix the sparse masks, and restart training with both LoRA and sparse adaptation (SpA) activated. All experiments, except for FFT, comfortably run on a single NVIDIA GeForce RTX 3090 GPU 24.3 GB memory (see Table 1).

5.2. Results

Main Experiment. In Table 1, we summarize our main experiments, which examine the accuracy of various fine-tuning approaches at various budgets across all the tasks considered. We consider three parameter budgets: 40 million, 80 million, and 160 million. For each budget, we explore five different ways of distributing parameters between LoRA and SpA, ranging from pure LoRA/SpA to intermediate sparse + low-rank budgets. The main experiments are conducted for a standard single pass over the dataset (epoch). However, for the smaller ViGGO and GSM8k datasets, we observe that extended training improves adaptation results. Hence, we also present the best results for each method from 2 and 3 epochs on these two datasets under the 'Extended' label. (We did not run extended training on SQL due to its much larger size.) Additionally, for QRoSA, we follow Dettmers et al. (2023a) and report the accuracy of the single-epoch adaptations when the pre-trained weights are 4-bit double-quantized.

Single-Pass Runs. The results in Table 1 show that, across all tasks and budgets, RoSA outperforms both LoRA and SpA. The only exception is the $80M$ budget trained on SQL, where LoRA marginally outperforms RoSA (89.1 vs 88.9). However, on the same task, RoSA $40M$ achieves a remarkable 89.7 accuracy. Surprisingly, in the single-epoch regime, RoSA even surpasses FFT significantly on all three datasets, highlighting the fast convergence of the hybrid adapter approach. This shows that this approach can be

particularly effective in the context of short, single-pass training, across tasks and parameter budgets.

Extended Training Experiments. The above conclusion still holds in extended experiments, where we find that RoSA can, in fact, match or even outperform FFT on both GSM8k (38.6% vs 38.8%) and ViGGO (97.3% vs 95.0%). Additionally, except for the $40M$ GSM8k, RoSA outperforms both LoRA and SpA. These results complement our single-pass experiments, indicating the superiority of RoSA in longer multiple-pass regimes. The fact that some of the best results for extended training are obtained on the medium-sized parameter budget suggests that the computational budget should be balanced against the active parameters for the run: the largest budget tends to yield the highest performance on the larger SQL dataset.

Overall, these results clearly highlight the effectiveness of RoSA; specifically, we find it remarkable that we are able to fully recover FFT accuracy while using parameter budgets that are 40-100x smaller. Finally, the memory overheads of maintaining sparse and low-rank components are indeed low: all our experiments fit inside a single 24GB GPU.

QRoSA: Quantizing Pre-trained Weights. Following QLoRA (Dettmers et al., 2023a), we repeat the single-pass experiments while double-quantizing the pre-trained weights to total memory. We observe that QRoSA slightly lags behind QLoRA in the larger budgets on the SQL dataset. However, it outperforms every other method (including FFT) on GSM8k by achieving 33.1 accuracy. Remarkably, in this setting, we need less than 12 GB of memory to match or exceed the accuracy of FFT on LLaMa2-7B!

Table 2: Comparison of fine-tuning LLaMA2-7B using different adaptation methods in terms of memory usage and accuracy on three datasets, while the pre-trained weights are 4-bit double-quantized following Dettmers et al. (2023a).

	Memory	GSM8k	ViGGO	SQL
FFT	> 60 GB	**32.3**	**82.1**	**89.0**
QLoRA $r = 16$	12.6 GB	29.8	88.0	88.2
QRoSA $r = 12, d = 0.15\%$	10.7 GB	**31.8**	93.8	88.5
QRoSA $r = 8, d = 0.3\%$	10.7 GB	30.9	**95.0**	**88.6**
QRoSA $r = 4, d = 0.45\%$	10.7 GB	30.3	92.4	86.7
QSpA $d = 0.6\%$	10.8 GB	22.8	89.5	79.2
QLoRA $r = 32$	13.0 GB	25.6	74.7	**89.0**
QRoSA $r = 24, d = 0.3\%$	11.0 GB	30.4	93.3	88.3
QRoSA $r = 16, d = 0.6\%$	11.1 GB	**33.1**	93.8	86.6
QRoSA $r = 8, d = 0.9\%$	11.1 GB	32.8	**95.4**	83.7
QSpA $d = 1.2\%$	11.3 GB	28.0	93.0	85.0
QLoRA $r = 64$	13.8 GB	30.6	88.1	**89.4**
QRoSA $r = 48, d = 0.6\%$	11.9 GB	30.5	93.6	81.6
QRoSA $r = 32, d = 1.2\%$	11.9 GB	**32.3**	94.3	88.2
QRoSA $r = 16, d = 1.8\%$	12.0 GB	30.8	**95.0**	88.5
QSpA $d = 2.4\%$	12.2 GB	28.9	90.8	42.9

Hyper-parameter Selection. Given a parameter budget, RoSA introduces a new hyper-parameter: the ratio by which we distribute the budget between the sparse and low-rank components. Our results in Table 1 show that in many cases there is a threshold for the LoRA rank above which the results do not improve further. The existence of this rank threshold was already known before, e.g., Section 7.2 in the original LoRA paper (Hu et al., 2021). In our experiments, this is more nuanced on the GSM8k and ViGGO datasets, where the optimal rank across different budgets is around 12-16, and the rest of the budget should be assigned to the sparse component to achieve the best results. This is justified by the fact that the difference between FFT and pre-trained weights has only a few large singular values (Figure 7). On the other hand, while hyper-parameter tuning is required to achieve the best results, we found that in almost all cases simply distributing the budget equally between the low-rank and sparse adapters is enough to outperform other adaptation methods. Hence distributing the budget half-half can serve as a solid default choice.

Mask Choice Ablation. We investigate the impact of different mask generation methods of RoSA for the GSM8k dataset in Table 3. Let $\tau_d(\cdot)$ be the TopK magnitude mask with density d. Then the methods we consider are:

1. *GradMag-LW (ours)*: $\mathcal{M} = \tau_d(\nabla \boldsymbol{W}^{+\tilde{L}})$
   A TopK magnitude mask on the accumulated square of gradients as described in Algorithm 1 following warm-up of the low-rank instance, where $\boldsymbol{W}^{+\tilde{L}} := \boldsymbol{W} + \tilde{\boldsymbol{\Delta}}_L$ and $\tilde{\boldsymbol{\Delta}}_L$ is the partially-trained low-rank instance.
2. *GradMag/GradFish*: $\mathcal{M} = \tau_d(\nabla \boldsymbol{W})$
   A TopK magnitude mask on gradients accumulated at initialization (in ℓ_1 or ℓ_2 norm squared), following FISH Mask (Sung et al., 2021).
3. *WeightRPCA*: $\mathcal{M} = \tau_d(\boldsymbol{W}_S)$
   The sparse component resulting from RPCA on the weights $\boldsymbol{W}$, $\boldsymbol{W}_S$, with a target density of d, following DSEE (Chen et al., 2021).
4. *GradRPCA*: $\mathcal{M} = \tau_d(\nabla \boldsymbol{W}_S)$
   The sparse component resulting from RPCA on the weight gradient $\nabla \boldsymbol{W}$, $\nabla \boldsymbol{W}_S$, with a target density of d, which we see as a natural combination of FISH Mask and DSEE.
5. *Lottery Ticket Update Masking (LTM)*: $\mathcal{M} = \tau_d(\boldsymbol{\Delta}^*_S)$
   For this, we try to identify a good set of coordinates to optimize over "in hindsight", by computing the sparse component of RPCA over the FFT update $\boldsymbol{\Delta}^*$, denoted by $\boldsymbol{\Delta}^*_S$, with a target density of d.
6. *RND(d)*: A random mask with density d.

First, we observe that the "Lottery Ticket" Mask (LTM), which has hindsight knowledge of the best optimization directions from the perspective of the FFT update, predictably

Table 3: Comparison of various masking methods: Training of LLaMa2-7B Model on GSM8k for 1 epoch using 80M trainable parameters.

Method	GSM8k Accuracy
LTM	**33.66**
GradMag-LW (ours)	**32.16**
GradMag (FISH Mask)	30.10
GradRPCA	29.87
WeightRPCA (DSEE)	30.71
RND	30.25

performs very well, being in fact competitive with FFT accuracy on GSM8k. The second best-performing method, by a significant margin, is given by the RoSA masks, coming within $\sim 1\%$ of the ideal mask. The remaining methods essentially perform within the variance of choosing random initial masks. The fact that gradient RPCA at initialization significantly under-performs our version suggests that the "warm-up" period is key to good accuracy. Overall, this suggests that choosing masks in a task-aware fashion is key to good performance in the context of LLM fine-tuning.

In summary, the experiments establish the fact that RoSA and QRoSA can indeed be competitive with the much more expensive FFT process in terms of top accuracy, while having a much lighter memory and computational footprint. This is enabled by our specific mask choice process, as well as by the efficient system support.

Runtime. Performing measurements on an NVIDIA RTX A6000 GPU, we find our current implementatio of RoSA to be approximately 1.7-2x slower than LoRA on the $80M$ parameter budget (see Appendix B). This is due to overheads in the sputnik implementation, which we plan to mitigate in future work. Furthermore, we note that fine-tuning on the down-stream tasks is usually a short process. Hence one can afford 1.7-2x slowdown compared to LoRA, considering that we are essentially able to recover FFT accuracy, and that FFT is usually either slower or not even executable in the memory-constrained setups we consider.

6. Discussion

In this paper, we took a step forward to address the problem of efficient fine-tuning of Large Language Models (LLMs). We proposed a method called Robust Adaptation (RoSA), which is inspired by the Robust PCA approach, and showed that RoSA significantly outperforms both low-rank adaptation (LoRA) (Hu et al., 2021) and prior sparse or hybrid approaches (Sung et al., 2021; Chen et al., 2021) at the same parameter budgets. Additionally, we came across the surprising observation that the best-performing RoSA can match or even outperform FFT in many settings. To complement our contributions, we provide an efficient PyTorch implementation of our method, aiming to make RoSA an accessible tool for researchers in the field.

Acknowledgments

The authors would like to thank Eldar Kurtic for experimental support and useful suggestions throughout the project.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Alex, N., Lifland, E., Tunstall, L., Thakur, A., Maham, P., Riedel, C. J., Hine, E., Ashurst, C., Sedille, P., Carlier, A., et al. Raft: A real-world few-shot text classification benchmark. *arXiv preprint arXiv:2109.14076*, 2021.

Askell, A., Bai, Y., Chen, A., Drain, D., Ganguli, D., Henighan, T., Jones, A., Joseph, N., Mann, B., DasSarma, N., et al. A general language assistant as a laboratory for alignment. *arXiv preprint arXiv:2112.00861*, 2021.

Bai, Y., Jones, A., Ndousse, K., Askell, A., Chen, A., DasSarma, N., Drain, D., Fort, S., Ganguli, D., Henighan, T., et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Candès, E. J., Li, X., Ma, Y., and Wright, J. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3): 1–37, 2011.

Castro, R. L., Ivanov, A., Andrade, D., Ben-Nun, T., Fraguela, B. B., and Hoefler, T. Venom: A vectorized n: M format for unleashing the power of sparse tensor cores. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14, 2023.

Chen, X., Chen, T., Chen, W., Awadallah, A. H., Wang, Z., and Cheng, Y. Dsee: Dually sparsity-embedded efficient tuning of pre-trained language models. *arXiv preprint arXiv:2111.00160*, 2021.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

De La Torre, F. and Black, M. J. A framework for robust subspace learning. *International Journal of Computer Vision*, 54:117–142, 2003.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. LLM.int8(): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022*, 2022.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023a.

Dettmers, T., Svirschevski, R., Egiazarian, V., Kuznedelev, D., Frantar, E., Ashkboos, S., Borzunov, A., Hoefler, T., and Alistarh, D. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*, 2023b.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Edalati, A., Tahaei, M., Kobyzev, I., Nia, V. P., Clark, J. J., and Rezagholizadeh, M. Krona: Parameter efficient tuning with kronecker adapter. *arXiv preprint arXiv:2212.10650*, 2022.

Evci, U., Gale, T., Menick, J., Castro, P. S., and Elsen, E. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pp. 2943–2952. PMLR, 2020.

Fischler, M. A. and Bolles, R. C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

Frantar, E. and Alistarh, D. Optimal brain compression: A framework for accurate post-training quantization and pruning. *Advances in Neural Information Processing Systems*, 35:4475–4488, 2022.

Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Gale, T., Zaharia, M., Young, C., and Elsen, E. Sparse GPU kernels for deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020*, 2020.

Gnanadesikan, R. and Kettenring, J. R. Robust estimates, residuals, and outlier detection with multiresponse data. *Biometrics*, pp. 81–124, 1972.

Gray, S., Radford, A., and Kingma, D. P. Gpu kernels for block-sparse weights. *arXiv preprint arXiv:1711.09224*, 3(2):2, 2017.

He, J., Zhou, C., Ma, X., Berg-Kirkpatrick, T., and Neubig, G. Towards a unified view of parameter-efficient transfer learning. In *Proceedings of the 10th International Conference on Learning Representations (ICLR-2022)*, 2022.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2020.

Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., and Peste, A. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22(1):10882–11005, 2021.

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

Hubara, I., Chmiel, B., Island, M., Banner, R., Naor, J., and Soudry, D. Accelerated sparse neural training: A provable and efficient method to find n: m transposable masks. *Advances in neural information processing systems*, 34: 21099–21111, 2021.

Huber, P. J. *Robust statistics*, volume 523. John Wiley & Sons, 2004.

Hyeon-Woo, N., Ye-Bin, M., and Oh, T.-H. Fedpara: Low-rank hadamard product for communication-efficient federated learning. *arXiv preprint arXiv:2108.06098*, 2021.

Ivanov, A., Dryden, N., and Hoefler, T. Sten: An interface for efficient sparsity in pytorch. 2022.

Jiang, P., Hu, L., and Song, S. Exposing and exploiting fine-grained block structures for fast and accurate sparse training. *Advances in Neural Information Processing Systems*, 35:38345–38357, 2022.

Juraska, J., Bowden, K., and Walker, M. ViGGO: A video game corpus for data-to-text generation in open-domain conversation. In *Proceedings of the 12th International Conference on Natural Language Generation*, pp. 164–172, Tokyo, Japan, October–November 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-8623. URL https://aclanthology.org/W19-8623.

Ke, Q. and Kanade, T. Robust l/sub 1/norm factorization in the presence of outliers and missing data by alternative convex programming. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pp. 739–746. IEEE, 2005.

Kurtic, E., Campos, D., Nguyen, T., Frantar, E., Kurtz, M., Fineran, B., Goin, M., and Alistarh, D. The optimal bert surgeon: Scalable and accurate second-order pruning for large language models. *arXiv preprint arXiv:2203.07259*, 2022.

Kurtic, E., Kuznedelev, D., Frantar, E., Goin, M., and Alistarh, D. Sparse finetuning for inference acceleration of large language models. *arXiv preprint arXiv:2310.06927*, 2023.

Lee, A. N., Hunter, C. J., and Ruiz, N. Platypus: Quick, cheap, and powerful refinement of llms. *arXiv preprint arXiv:2308.07317*, 2023.

Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

Li, S., Osawa, K., and Hoefler, T. Efficient quantized sparse matrix operations on tensor cores. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15. IEEE, 2022.

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

Li, Y., Yu, Y., Liang, C., He, P., Karampatziakis, N., Chen, W., and Zhao, T. Loftq: Lora-fine-tuning-aware quantization for large language models. *arXiv preprint arXiv:2310.08659*, 2023.

Liu, H., Tam, D., Muqeeth, M., Mohta, J., Huang, T., Bansal, M., and Raffel, C. A. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35: 1950–1965, 2022.

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

Liu, X., Zheng, Y., Du, Z., Ding, M., Qian, Y., Yang, Z., and Tang, J. Gpt understands, too. *AI Open*, 2023.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Mangrulkar, S., Gugger, S., Debut, L., Belkada, Y., Paul, S., and Bossan, B. Peft: State-of-the-art parameter-efficient fine-tuning methods. `https://github.com/huggingface/peft`, 2022.

Min, S., Lewis, M., Zettlemoyer, L., and Hajishirzi, H. Metaicl: Learning to learn in context. *arXiv preprint arXiv:2110.15943*, 2021.

MosaicML. LLM Foundry, 2023a. URL `https://github.com/mosaicml/llm-foundry`.

MosaicML. Introducing mpt-7b: A new standard for open-source, commercially usable llms, 2023b. URL `www.mosaicml.com/blog/mpt-7b`. Accessed: 2023-12-22.

Niederfahrenhorst, A., Hakhamaneshi, K., and Ahmad, R. Fine-Tuning LLMs: LoRA or Full-Parameter?, 2023. URL `https://www.anyscale.com/blog/fine-tuning-llms-lora-or-full-parameter-an-in-depth-analysis-with-llama-2`.

Nikdan, M., Pegolotti, T., Iofinova, E., Kurtic, E., and Alistarh, D. Sparseprop: Efficient sparse backpropagation for faster training of neural networks at the edge. In *International Conference on Machine Learning*, pp. 26215–26227. PMLR, 2023.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

Peste, A., Iofinova, E., Vladu, A., and Alistarh, D. Ac/dc: Alternating compressed/decompressed training of deep neural networks. *Advances in neural information processing systems*, 34:8557–8570, 2021.

Qiu, Z., Liu, W., Feng, H., Xue, Y., Feng, Y., Liu, Z., Zhang, D., Weller, A., and Schölkopf, B. Controlling text-to-image diffusion by orthogonal finetuning. *arXiv preprint arXiv:2306.07280*, 2023.

Sanh, V., Wolf, T., and Rush, A. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in Neural Information Processing Systems*, 33:20378–20389, 2020.

Sanh, V., Webson, A., Raffel, C., Bach, S. H., Sutawika, L., Alyafeai, Z., Chaffin, A., Stiegler, A., Scao, T. L., Raja, A., et al. Multitask prompted training enables zero-shot task generalization. *arXiv preprint arXiv:2110.08207*, 2021.

Singh, S. P. and Alistarh, D. Woodfisher: Efficient second-order approximation for neural network compression. *Advances in Neural Information Processing Systems*, 33: 18098–18109, 2020.

Sung, Y.-L., Nair, V., and Raffel, C. A. Training neural networks with fixed sparse masks. *Advances in Neural Information Processing Systems*, 34:24193–24205, 2021.

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

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

Wang, Y., Kordi, Y., Mishra, S., Liu, A., Smith, N. A., Khashabi, D., and Hajishirzi, H. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*, 2022a.

Wang, Y., Mishra, S., Alipoormolabashi, P., Kordi, Y., Mirzaei, A., Naik, A., Ashok, A., Dhanasekaran, A. S., Arunkumar, A., Stap, D., et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 5085–5109, 2022b.

Wei, J., Bosma, M., Zhao, V. Y., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

Wright, J., Ganesh, A., Rao, S., Peng, Y., and Ma, Y. Robust principal component analysis: Exact recovery of corrupted low-rank matrices via convex optimization. *Advances in neural information processing systems*, 22, 2009.

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.

Zhang, Q., Chen, M., Bukharin, A., He, P., Cheng, Y., Chen, W., and Zhao, T. Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*, 2023.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Zhong, V., Xiong, C., and Socher, R. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017.

Zhou, T. and Tao, D. Greedy bilateral sketch, completion & smoothing. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, volume 31 of *Proceedings of Machine Learning Research*, pp. 650–658. PMLR, 2013.

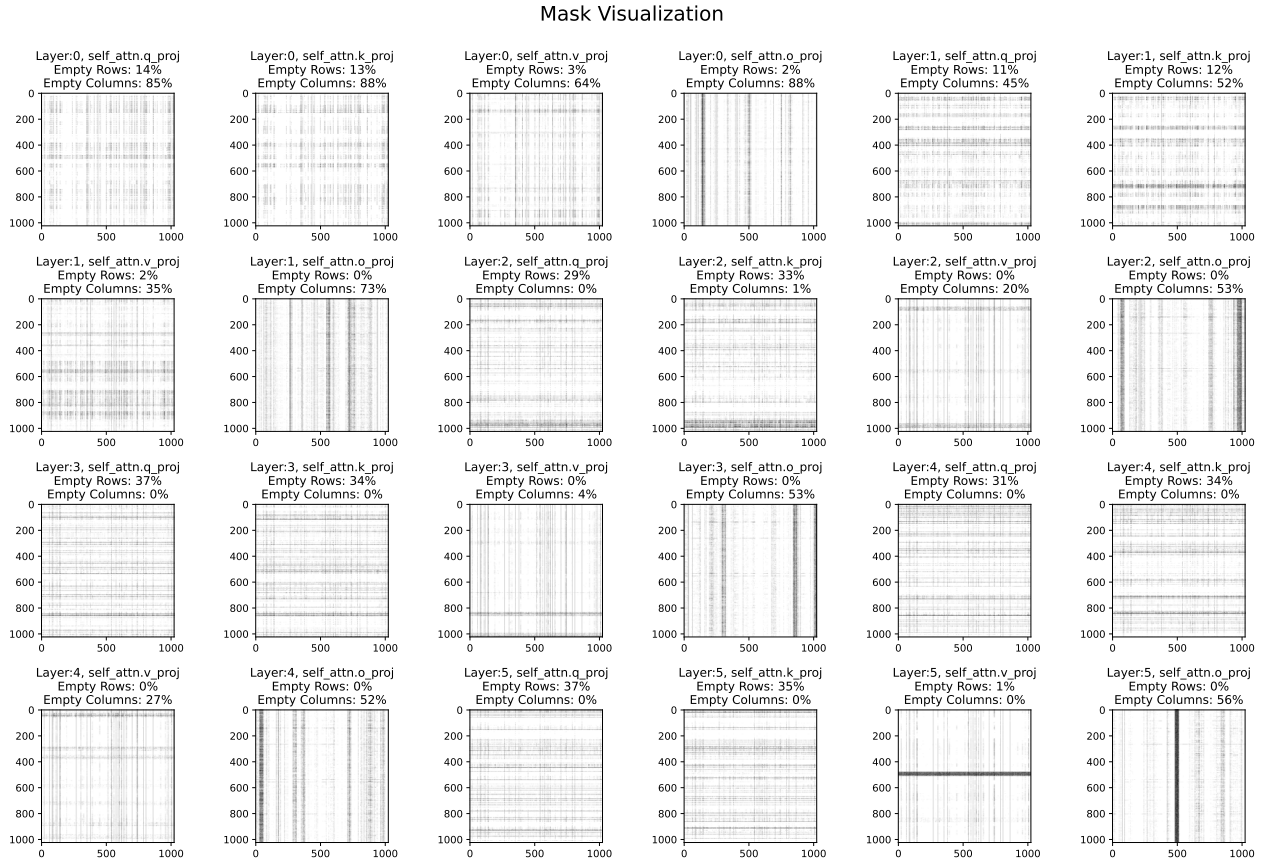


Figure 5: Here we see a visualization of a subset of masks taken from LLaMa2-7B Model trained on GSM8k ($r = 16, d = 0.6\%$). We can see that most masks visualized here have either a significant number of empty rows or columns. For the purposes of visualization, each mask is max-pooled with a kernel size and stride of 4.

A. System Details

We integrated RoSA into a fork of the standard `peft` library (Mangrulkar et al., 2022), and performed all the experiments using the the `llm-foundry` codebase (MosaicML, 2023a). Next, we will elaborate on the efficient implementation of RoSA.

Mask Structure. As noted in Section 4, our findings show that a significant number of either mask rows or columns are completely empty. Figure 5 shows a visualization of this phenomenon, and Table 4 outlines the empty rows across a wider range of models over a subset of our models. It shows, for each model, the mean of the maximum percentage of empty rows or columns. Finally, we report that a mean of 46.74% (rounded to two decimals) of the maximum between the percentage of empty rows or columns is present across all of our trained models. The prevalence of empty rows and columns emphasizes the motivation to use a kernel that does not launch threads for outputs where no work is needed.

A.1. SDDMM Kernel

Our SDDMM kernel is based on the `sputnik` kernel (Gale et al., 2020). Their original SDDMM implementation was extended in two ways. First, the original SDDMM kernel, as noted in the referenced publication, launches the maximum number of threads over the entire output matrix and then simply terminates those threads that have no work to do. In order to accommodate the fact that a significant portion of either the rows or columns of each individual mask is empty, we limit the number of threads launched to the number of rows and columns that have a non-zero value. At first glance, this seems to contradict the original paper's claim that the extra threads don't induce significant overhead. However, the original publication did not focus on benchmarking the low sparsity and structures present in this paper. Furthermore, as row sorting

Table 4: This table shows the row and column statistics for a subset of the models with across a wide range of datasets and densities. Note that the masks depend on the learning rate because they were generated after a LoRA warmup period.

LLaMA 7B	Maximal Empty Row	Maximal Empty Column	Mean Maximal Empty Row or Column
GSM8K			
$d = 0.0015, r = 12, lr = 0.0002$	98.18%	98.97%	73.49%
$d = 0.003, r = 8, lr = 0.0002$	97.85%	96.72%	58.30%
$d = 0.006, r = 48, lr = 0.0002$	97.5%	94.03%	40.94%
$d = 0.012, r = 32, lr = 0.0002$	96.46%	85.01%	27.12%
$d = 0.018, r = 16, lr = 0.0004$	94.79%	79.94%	19.60%
SQL			
$d = 0.0015, r = 12, lr = 0.0004$	99.14%	97.92%	79.34%
$d = 0.003, r = 8, lr = 0.0004$	98.61%	96.72%	65.94%
$d = 0.0045, r = 4, lr = 0.0004$	97.96%	95.70%	56.36%
$d = 0.006, r = 48, lr = 0.0004$	96.56%	94.10%	48.84%
$d = 0.009, r = 8, lr = 0.0001$	95.32%	87.28%	41.25%
$d = 0.012, r = 32, lr = 0.0004$	91.13%	85.15%	34.46%
$d = 0.018, r = 16, lr = 0.0004$	86.87%	80.06%	29.74%
ViGGO			
$d = 0.0015, r = 12, lr = 0.0002$	99.53%	98.90%	75.29%
$d = 0.003, r = 8, lr = 0.0002$	99.04%	97.50%	61.68%
$d = 0.0045, r = 4, lr = 0.0002$	96.19%	91.43%	55.22%
$d = 0.006, r = 48, lr = 0.0002$	91.38%	90.91%	46.14%
$d = 0.009, r = 8, lr = 0.0002$	95.27%	92.19%	37.95%
$d = 0.012, r = 32, lr = 0.0002$	94.28%	87.32%	30.88%
$d = 0.018, r = 16, lr = 0.0002$	92.11%	82.88%	24.11%

according to the number of non-zero values is part of the original implementation's pipeline, the additional necessary kernel launch information can be calculated without significant overhead. Second, the SDDMM implementation was extended to support 16-bit indices.

We present the benchmark results of these two changes in Figure 6. We extract masks from LLaMA2-7B $d = 0.6\%$ and $r = 16$. For each mask $\boldsymbol{M}$ and construct two randomly generated `float32` matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ with dimensions (M, K) and (N, K) and compute the SDDMM. We have a fixed $K = 512$ in this synthetic benchmark. The durations are rounded to two decimal places.

A.2. CSR-ADD Kernel

A CUDA kernel calculating the $\boldsymbol{A} = \boldsymbol{A} + \boldsymbol{B}$ operation where $\boldsymbol{A}$ is dense and $\boldsymbol{B}$ is sparse (stored in the CSR format), was implemented with support for `float32`, `float16` and `bfloat16` input data types. It distributes thread blocks over rows of $\boldsymbol{B}$ with each warp, then goes over the nonzero values and adds them to the dense matrix.

A.3. Other Details

RoSA Pseudocode. We include a straight-forward pseudocode that describes our adaptation method (Algorithm 2).

Gradient Collection for QRoSA. Since automatic differentiation is not supported for quantized tensors in PyTorch, in the QRoSA experiments, we manually multiply the output gradients and inputs during training to calculate the weight gradients required for mask collection.

B. Runtime

In Table 5 we compare the runtime of RoSA and LoRA on an NVIDIA RTX A6000. We observe a slow-down relative to LoRA of around 2x. This is because of overheads due to sparsity, but also because the sparse operators we use work with FP32 precision, which is slower than LoRA operations, which employ FP16.

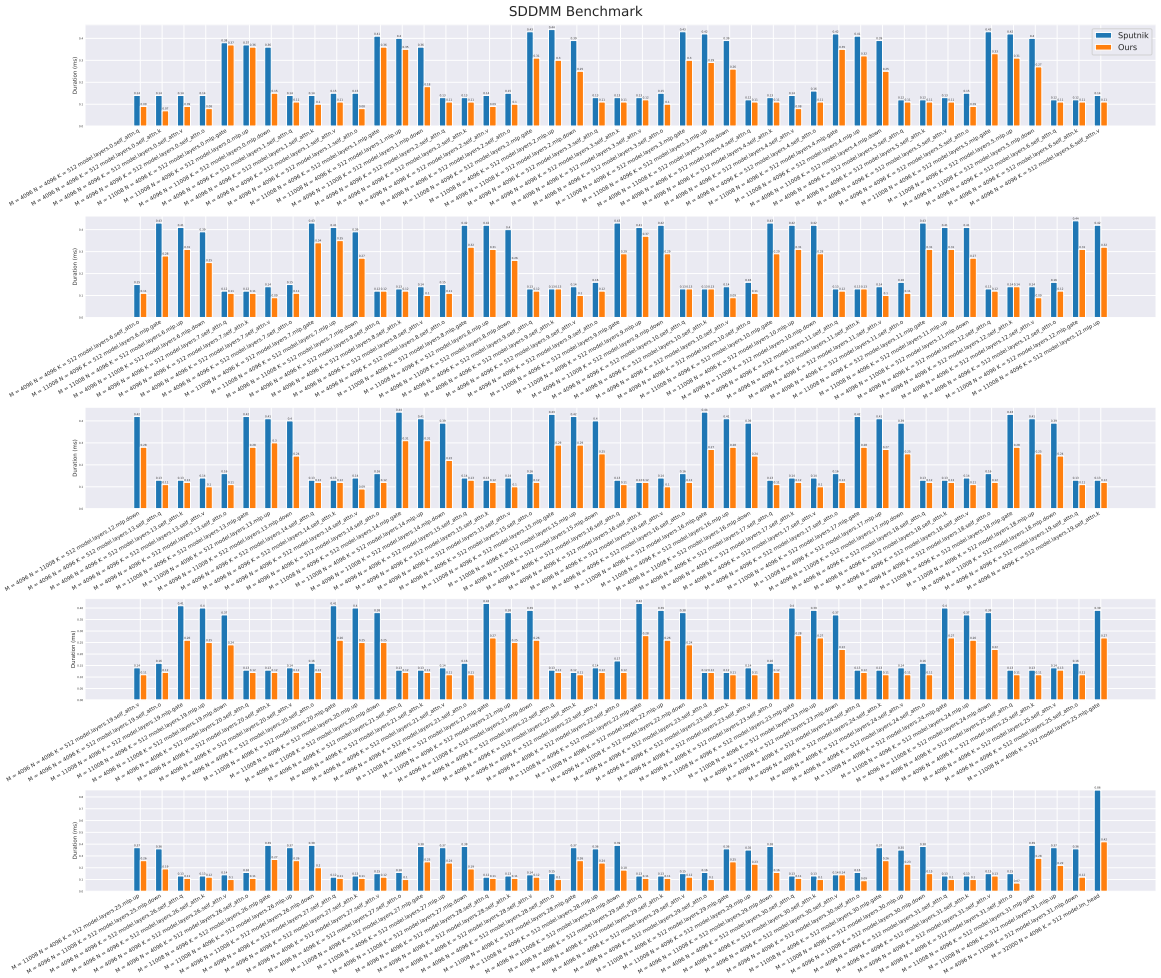


Figure 6: This figure shows the result of benchmarking SDDMM kernels with masks extracted from LLaMA2-7B $d = 0.6\%$ and $r = 16$. Compared to `sputnik` we achieve a geometric mean speedup of 1.36x and a peak speedup of 3x.

C. Comparison with IA3

In this section, we compare our proposed method, RoSA, with IA3 (Liu et al., 2022), another parameter-efficient fine-tuning technique. IA3 involves introducing scaling parameters for the activations within a neural network.

Table 6 shows that IA3 performs poorly compared to RoSA and LoRA in terms of accuracy on the three GSM8k, ViGGO, and SQL datasets. One explanation for this underperformance is that IA3 clearly underfits due to its small parameter count. Unlike RoSA and LoRA, which introduce additional parameters through low-rank and sparse adaptations, IA3's scaling parameters are insufficient to capture the complexity of the tasks, leading to suboptimal performance.

However, it is important to note that IA3 is designed to excel in few-shot learning scenarios. For example, on the RAFT dataset (Alex et al., 2021), which is specifically curated for few-shot learning tasks, IA3 demonstrates competitive performance. This is in contrast to RoSA and LoRA, which generally require a larger dataset to achieve optimal results.

Algorithm 2 Robust Adaptation (RoSA)

Require: $\mathcal{W}, \bar{w} \leftarrow$ the fully connected weights and the rest of the LLM parameters, respectively
Require: $\mathcal{D} \leftarrow$ the downstream dataset
Require: $\mathcal{L}(.) \leftarrow$ the loss function
Require: $r \leftarrow$ LoRA rank
Require: $d \leftarrow$ SpA density
Require: $m \leftarrow$ number of samples to use for mask generation
`[m random samples for mask generation]`
$\mathcal{D}_{\mathcal{M}} \leftarrow$ *random-subset*$(\mathcal{D}, m)$
`[run Algorithm 1 to generate the masks]`
$\mathcal{M} \leftarrow$ *generate-masks*$(\mathcal{W}, \bar{w}, \mathcal{D}_{\mathcal{M}}, \mathcal{L}, d)$
$k \leftarrow$ *length*$(\mathcal{W})$
for $i \in \{1, 2, ..., k\}$ **do**
  $m_i, n_i \leftarrow$ *shape*$(\boldsymbol{W}_i)$
  `[init LoRA ((Hu et al., 2021))]`
  $\boldsymbol{\Delta}_i^L \leftarrow$ *initialize-lora-params*(m_i, n_i, r)
  `[init SpA with zero]`
  $\boldsymbol{\Delta}_i^S \leftarrow$ *initialize-spa-params*$(\mathcal{M}_i)$
end for
$\Delta^L \leftarrow \{\boldsymbol{\Delta}_1^L, \boldsymbol{\Delta}_2^L, ..., \boldsymbol{\Delta}_k^L\}$
$\Delta^S \leftarrow \{\boldsymbol{\Delta}_1^S, \boldsymbol{\Delta}_2^S, ..., \boldsymbol{\Delta}_k^S\}$
`[train the adapters]`
$\Delta_*^L, \Delta_*^S \leftarrow \underset{\Delta^L, \Delta^S}{\arg\min}\, \mathcal{L}(\mathcal{D}; \mathcal{W} + \Delta^L + \Delta^S, \bar{w})$
return Δ_*^L, Δ_*^S

Table 5: Runtime comparison between (Q)LoRA and (Q)RoSA in the 80M parameter budget. The measurements are done using an NVIDIA RTX A6000 GPU.

Method	batch/second
LoRA $r = 32$	0.1149
RoSA $r = 24, d = 0.3\%$	0.0602
RoSA $r = 16, d = 0.6\%$	0.0595
RoSA $r = 8, d = 0.9\%$	0.0575
SpA $d = 1.2\%$	0.0622
QLoRA $r = 32$	0.0911
QRoSA $r = 24, d = 0.3\%$	0.0531
QRoSA $r = 16, d = 0.6\%$	0.0521
QRoSA $r = 8, d = 0.9\%$	0.0515
QSpA $d = 1.2\%$	0.0546

D. Singular Value Analysis on Full Fine-Tuning

We present a straightforward analysis of the singular values obtained from $\boldsymbol{\Delta}^*$ of the LLaMA2-7B model (Touvron et al., 2023b) fine-tuned on the GSM8k dataset. The focus is on a set of plots representing singular values from several randomly selected layers of the LLaMA2-7B model. The plots in Figure 7 reveal a notable pattern: a few singular values are significantly larger compared to the rest, which is relatively small yet not zero.

This pattern in the singular values suggests that the updates made during full fine-tuning of LLaMA2 exhibit a tendency towards a low-rank structure. However, they cannot be considered purely low-rank due to the presence of these small, non-zero singular values.

E. Instruction-tuning Results

In this section, we present our findings from training the LLaMA2-7B model on the OpenPlatypus and Alpaca datasets. The OpenPlatypus dataset (Lee et al., 2023), and the Alpaca dataset (Taori et al., 2023), are both designed to enhance instruction-following capabilities in language models. To evaluate the performance of our method, we report the accuracy

Table 6: Comparison of fine-tuning LLaMA2-7B using FFT, RoSA and IA3 (Liu et al., 2022). For RoSA, we consider 40M, 80M, and 160M parameter budgets and we assume the budget is distributed equally between the sparse and low-rank adapters.

		GSM8k		ViGGO		SQL
	#Params	1 Epoch	Extended	1 Epoch	Extended	1 Epoch
FFT	6.7 B	**32.3**	**38.8**	**82.1**	**95.0**	**89.0**
RoSA $r = 8, d = 0.3\%$	40.8 M	29.2	37.5	94.5	97.1	77.6
RoSA $r = 16, d = 0.6\%$	81.6 M	**32.2**	**38.6**	**95.2**	97.1	88.3
RoSA $r = 32, d = 1.2\%$	163.1 M	**32.2**	36.2	93.4	**97.3**	**89.2**
IA3	1.6 M	13.12	16.07	38.24	40.06	84.5

on the Massive Multitask Language Understanding (MMLU) benchmark (Hendrycks et al., 2020), a comprehensive suite designed to test models across a wide range of academic and professional subjects.

Results. Table 7 summarizes our results. Our experiments reveal that RoSA does not consistently outperform LoRA on instruction-tuning, particularly when tuning on datasets such as OpenPlatypus and Alpaca, which contain data relatively similar to the pre-training data.

As discussed earlier in the paper (refer to Section 3.2), the advantage of RoSA is more pronounced when the training data is rather complex, i.e. in settings where full fine-tuning significantly outperforms LoRA. This observation aligns with our current results, suggesting that for simpler instruction tuning tasks, LoRA performs adequately, matching or even outperforming FFT, and therefore RoSA is not necessary.

Analysis. The primary reason for RoSA's underperformance in these scenarios might be that, as mentioned earlier in the paper, when the tasks are not complex enough, RoSA's performance is on par with LoRA's. Another reason is based on the findings from He et al. (2022), which indicate that added parameters are better utilized in the feed-forward network (FFN) layers rather than the attention layers. Since RoSA is more robust at capturing complex information, it is more effective when the added parameters are used in the MLP modules. Therefore, exploring different choices of target fine-tuning modules might be better to yield better performance; however, we leave this for further research.

Table 7: MMLU 5-shot accuracy comparison on LLaMA2-7B. We observe that RoSA does not outperform LoRA on these simpler tasks.

Dataset	Model/Fine-tuning	MMLU (5-shot)
	LLaMa2-7B	45.75
OpenPlatypus	LoRA $r = 16$	49.92
	RoSA $r = 16, d = 0.6\%$	46.54
Alpaca	LoRA $r = 16$	45.8
	RoSA $r = 16, d = 0.6\%$	46.52

F. Qualitative Results

Here we showcase some examples of the generated outputs on the GSM8k task, which includes mathematical reasoning. We consider the FFT model, as well as LoRA, SpA, and best RoSA models in the $40M$ budget (single epoch).

Question:

```
John rents his car out 10 times a month for 3 hours each time.  He
gets paid $25 an hour.  If his car payment is $500, how much profit
does he make on his car?
```

Answer (Ground Truth):

```
He rents his car 10*3=«10*3=30»30 hour a month
So he makes 25*30=$«25*30=750»750 a month
That means he has a profit of 750-500=$«750-500=250»250 a month
```

```
#### 250
```

Answer (FFT):

```
He rents his car out 10*3=«10*3=30»30 hours a month
So he makes 30*25=$«30*25=750»750 a month
So his profit is 750-500=$«750-500=250»250
#### 250
```

Answer (LoRA):

```
He makes 3*10=$«3*10=30»30 per month
So he makes 30*12=$«30*12=360»360 per year
So he makes 360-500=$«360-500=140»140 profit
#### 140
```

Answer (SpA):

```
He makes 3*10=$«3*10=30»30 per month
So he makes 30*12=$«30*12=360»360 per year
So he makes a profit of 360-500=$«360-500=140»140
#### 140
```

Answer (RoSA):

```
He makes 3*10=$«3*10=30»30 per month
So he makes 30*25=$«30*25=750»750 per month
So he makes 750-500=$«750-500=250»250 per month
#### 250
```

In the above example, while both LoRA and SpA decide to calculate yearly profit, RoSA successfully solves the problem correctly.

Question:

```
Mary is two years younger than Joan, who is five years older than
Jessa.  If Jessa is 20 years old, what is the sum of the ages of the
three girls?
```

Answer (Ground Truth):

```
Joan is 20 + 5 = «20+5=25»25 years old.
Mary is 25 - 2 = «25-2=23»23 years old.
Therefore, the sum of their ages is 20 + 25 + 23 = «20+25+23=68»68.
#### 68
```

Answer (FFT):

```
Joan is 20 + 5 = «20+5=25»25 years old.
Mary is 25 - 2 = «25-2=23»23 years old.
The sum of the ages of the three girls is 23 + 20 + 25 =
«23+20+25=68»68 years old.
#### 68
```

Answer (LoRA):

```
The sum of the ages of the three girls is 20 + 5 + 2 = «20+5+2=27»27
years.
#### 27
```

Answer (SpA):

```
Mary is 20 - 5 = «20-5=15»15 years old.
The sum of the ages of the three girls is 20 + 15 + 15 =
«20+15+15=50»50 years.
#### 50
```

Answer (RoSA):

```
Joan is 20 + 5 = «20+5=25»25 years old.
Mary is 25 - 2 = «25-2=23»23 years old.
The sum of the ages of the three girls is 23 + 20 + 20 =
«23+20+20=63»63 years.
#### 63
```

While all adaptation methods (including RoSA) fail to answer the question correctly, we see that LoRA and SpA completely fail to even process it. In contrast, RoSA calculates the ages correctly and only fails to sum them up at the end.

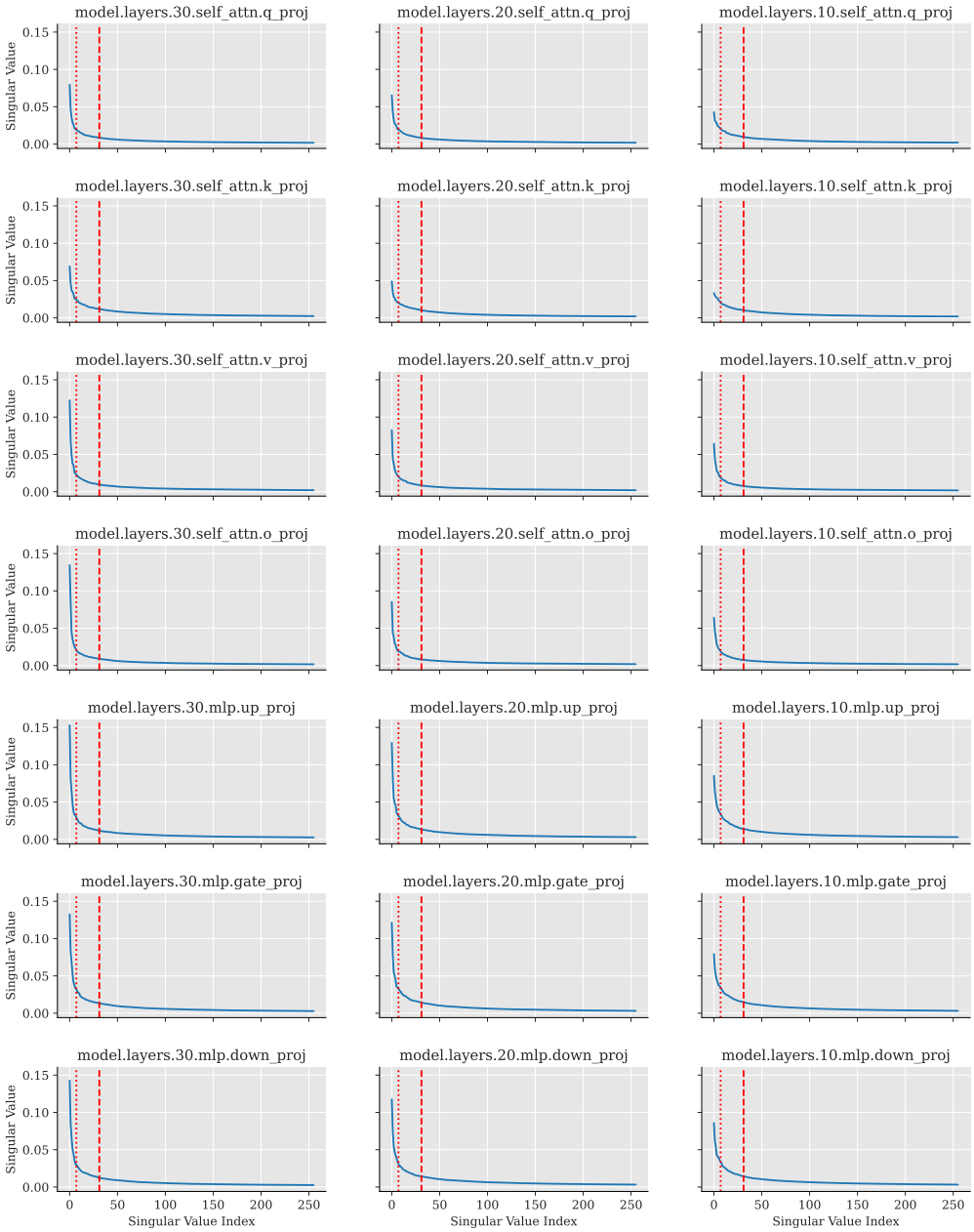


Figure 7: Sorted singular values of Δ^* for various layers of a LLaMA2-7B fully fine-tuned on GSM8k. Thresholds for ranks 8 and 32 are marked with dotted and dashed lines, respectively. The top 256 singular values are selected.