

# Step-by-Step Unmasking for Parameter-Efficient Fine-tuning of Large Language Models

Aradhya Agarwal\* Suhas Kamasetty Ramesh\* Ayan Sengupta\* Tanmoy Chakraborty

Indian Institute of Technology Delhi, India

Aradhya.Agarwal.cs520@cse.iitd.ac.in, suhaskr@gmail.com  
ayan.sengupta@ee.iitd.ac.in, tanchak@iitd.ac.in

## Abstract

Fine-tuning large language models (LLMs) on downstream tasks requires substantial computational resources. A class of parameter-efficient fine-tuning (PEFT) aims to mitigate these computational challenges by selectively fine-tuning only a small fraction of the model parameters. Although computationally efficient, these techniques often fail to match the performance of fully fine-tuned models, primarily due to inherent biases introduced during parameter selection. Traditional selective PEFT techniques use a fixed set of parameters based on a pre-defined budget (a process also known as *unmasking*), failing to capture parameter importance dynamically and often ending up exceeding the budget. We introduce  $\mathbf{ID}^3$ , a novel selective PEFT method that calculates parameter importance continually and dynamically unmask parameters by balancing exploration and exploitation in parameter selection. Our empirical study on 15 tasks spanning natural language understanding and generative tasks demonstrates the effectiveness of our method compared to fixed-masking-based PEFT techniques. We analytically show that  $\mathbf{ID}^3$  reduces the number of gradient updates by a factor of two, enhancing computational efficiency.  $\mathbf{ID}^3$  is robust to random initialization of neurons and, therefore, can be seamlessly integrated into existing additive and reparametrization-based PEFT modules such as adapters and LoRA for dynamic sparsification.<sup>1</sup>

## 1 Introduction

Pre-trained large language models (Devlin et al., 2018; Liu et al., 2019; Raffel et al., 2020; Brown et al., 2020; Touvron et al., 2023) have demonstrated remarkable capabilities in understanding

and generating natural language. To adapt these models for downstream tasks, fine-tuning on task-specific datasets is essential in order for the models to acquire specialized knowledge relevant to particular tasks and domains. Larger pre-trained models, such as GPT-3 (Brown et al., 2020) and Llama (Touvron et al., 2023), exhibit even more advanced language understanding and reasoning skills, which enable them to leverage emergent capabilities (Radford et al., 2019) like in-context learning (ICL). This ability allows these models to quickly adapt to new tasks without needing gradient-based training. However, recent research (Liu et al., 2022) indicates that, despite its practical advantages, ICL is often less efficient than fine-tuning in terms of computational cost and downstream performance. Consequently, parameter-efficient fine-tuning has become increasingly important for effectively adapting large pre-trained models to specific tasks.

Parameter-efficient fine-tuning (PEFT) aims to increase the memory and computational efficiency of model fine-tuning by performing low-rank or sparse updates instead of dense updates, as is typical in the case of full fine-tuning (FFT). Additive PEFT methods (Houlsby et al., 2019; Pfeiffer et al., 2020) introduce additional trainable parameters to the frozen pre-trained model, whereas reparametrization-based PEFT techniques (Hu et al., 2021; He et al., 2022) utilize low-rank representations of existing model parameters to reduce the number of trainable parameters. Selective methods (Liao et al., 2023; Sung et al., 2021; Zaken et al., 2021), another class of PEFT techniques, use different heuristics to select the parameters within the pre-trained models for fine-tuning. The heuristic function assigns positive real-valued importance to each parameter in the model, while a suitable selection strategy determines which parameters to retain for fine-tuning. For instance, Diff Pruning (Guo et al., 2020) uses

\*Equal contribution

<sup>1</sup>Code is available at <https://github.com/Aradhya2002/selective-peft-toolkit>

change in parameter magnitude to assess the parameter importance, whereas, Fish mask (Sung et al., 2021) uses gradient-based Fisher importance heuristic function. A majority of these selective PEFT techniques identify and fine-tune only a static set of top- $B$  parameters (*aka* ‘budget’) from the entire parameter pool, with a fixed and predefined budget  $B$ . Incorrect allocation of this budget can detrimentally impact the model’s fine-tuning performance due to the misselection of parameters, either by including non-essential ones or excluding critical ones. The parameter selection strategies for these PEFT techniques can be broadly classified into two classes – static (static- $\mathcal{S}$ ) and repeated (repeat- $\mathcal{S}$ ). These classes utilize two extreme cases of exploration and exploitation of parameters with static- $\mathcal{S}$  as exploitation-only (*i.e.*, reuse same parameters), while repeat- $\mathcal{S}$  as exploration-only (*i.e.*, always choose new parameters). A majority of the existing selective PEFT methods use static- $\mathcal{S}$  selection, and these exploitation-only methods often fail to select the optimal parameters for a given task and instead reuse possibly redundant features. On the other hand, repeat- $\mathcal{S}$ -based PEFT methods often overshoot the target budget and perform well only for very small budgets. Selective PEFT methods like Diff Pruning and Fish Mask often use masking matrices but fail to utilize their sparsity, leading to computational costs akin to full fine-tuning.

To address these issues, we introduce a novel selection strategy increment- $\mathcal{S}$ , which balances the exploration and exploitation strategies adopted in repeat- $\mathcal{S}$  and static- $\mathcal{S}$ . We analytically show that incremental parameter selection is computationally more efficient and also practically beneficial as it provides fine-grained control over the budget, unlike existing methods. Moreover, we experimentally show that despite performing half the number of gradient updates, the performance with increment- $\mathcal{S}$  exceeds existing baselines. We also propose a new **D**ynamic **m**agnitu**D**e and **g**ra**D**ient-based heuristic (*aka*  $\mathbf{D}^3$ ), which combines the benefits of magnitude and gradient-based parameter importance heuristics. Our proposed method, which we call increment- $\mathbf{D}^3$  (*aka*  $\mathbf{ID}^3$ ), can be easily integrated into any neural module and sparsify additive and reparameterized modules of pre-trained models. Existing static- $\mathcal{S}$  PEFT techniques do not exhibit this property, as they fail to assess parameter importance for randomly initial-

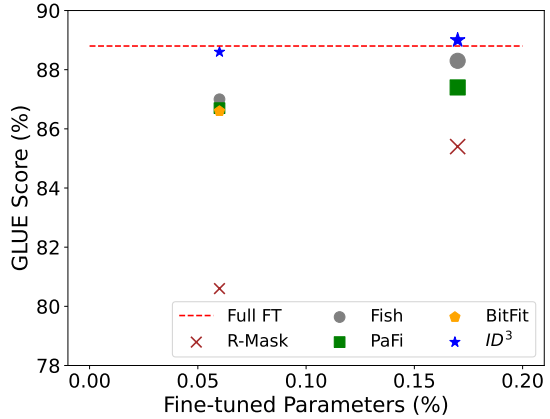


Figure 1: Comparison of different selective PEFT methods – Full fine-tuning (Full FT), Random masking (R-Mask), Fish (Sung et al., 2021), BitFit (Zaken et al., 2021), PaFi (Liao et al., 2023) and  $\mathbf{ID}^3$  on GLUE benchmark. Marker size denotes the number of trainable parameters. Detailed results are reported in Table 1.

ized untrained parameters.

We evaluate the effectiveness of various selective PEFT methods on the GLUE benchmark (Wang et al., 2018) comprising eight natural language understanding tasks. For a budget of 103K,  $\mathbf{ID}^3$  outperforms the other selective PEFT baselines by a wide margin of 1.2% with the pre-trained DeBERTa-v3 (He et al., 2021b) backbone. With only 0.17% of trainable parameters,  $\mathbf{ID}^3$  beats the fully fine-tuned DeBERTa-v3 model with a margin of 0.3% across all GLUE tasks (c.f. Figure 1). We further explore  $\mathbf{ID}^3$  with LoRA (Hu et al., 2021) on a pre-trained Llama-7b (Touvron et al., 2023) backbone model on six mathematical reasoning tasks. With 94% sparsity,  $\mathbf{ID}^3$  achieves 0.5% better accuracy on mathematical reasoning tasks than LoRA, emphasizing the usefulness of  $\mathbf{ID}^3$  on untrained (randomly initialized) components.

Our major contributions are listed below:

- (1) We introduce a novel selective strategy, increment- $\mathcal{S}$ , for parameter-efficient fine-tuning, which enables incremental parameter selection and dynamic assessment of parameter importance.
- (2) We propose a new importance-based heuristic,  $\mathbf{D}^3$ , that combines the benefits of gradient and magnitude-based parameter importance functions. Together with increment- $\mathcal{S}$  strategy, our proposed selective PEFT method  $\mathbf{ID}^3$  demonstrates a strong performance on various nat-

ural language understanding and generation tasks, even with highly sparse parameter updation.

- (3) Our method produces a series of progressively improved models across various budget levels, allowing users to balance budget and performance effectively.
- (4) We provide an open-source toolkit integrating four selective PEFT techniques, offering comprehensive support for selective methods that is not available in existing toolkits.

## 2 Related Work

This section highlights the representative works in three broad categories of PEFT strategies – *additive*, *reparameterized* and *selective*.

Additive PEFT methods, such as Adapters (Houlsby et al., 2019; Pfeiffer et al., 2020), add additional neural components to the pre-trained models. Due to their additive nature, these methodologies usually offer flexibility in multi-task fine-tuning setups, where the same pre-trained model is used with different task-specific adapters. The earliest adapter technique proposed by (Houlsby et al., 2019) utilized the additive component in feed-forward networks of self-attention (Vaswani et al., 2017). Subsequent additive PEFT methods (He et al., 2021a; Li and Liang, 2021; Zhu et al., 2021) differ in terms of placement of these additive components. The reparameterization-based PEFT techniques such as LoRA (Hu et al., 2021) use a low-rank approximation of the parameter update matrix  $\Delta W = BA$  to reduce the effective number of trainable parameters. Meanwhile, LoRA applies a uniform rank across all incremental parameters, assuming that all parameter matrices are equally important. To address this limitation, AdaLoRA (Zhang et al., 2023b) dynamically allocates the parameter budget among the additional weight matrices with singular value decomposition of the  $\Delta W$  and importance-aware rank allocation. IncreLoRA (Zhang et al., 2023a) proposed an incremental parameter allocation method that computes the importance scores of each module and adaptively adds the most important components to the trainable parameters.

Selective parameter-efficient fine-tuning strategies generate a sparse mask  $M \in \{0, 1\}^{|W|}$  corresponding to each weight matrix  $W$  in the pre-trained model. Unlike additive and

reparameterization-based techniques, selective methods consider the importance of individual parameters instead of the entire component. In this context, BitFit (Zaken et al., 2021) selectively trains the bias terms within each model unit. In contrast, Diff pruning (Guo et al., 2020) evaluates the absolute parameter changes across successive training phases, pruning those with the smallest magnitude. The computation of the magnitude of change of parameters requires significant computational and storage costs, equivalent to full fine-tuning of the model. To alleviate these computational burdens, Sung et al. (2021) proposed Fish Mask that computes fixed sparse masks with an empirical Fisher importance matrix. To avoid the computation cost of the parameter importance, Liao et al. (2023) proposed PaFi, which assesses the significance based on the absolute magnitude of the parameters and prunes the unimportant ones. Unlike earlier methods that modify the pre-trained model directly, He et al. (2022) proposed SparseAdapter, a novel approach that merges with existing adapter-based techniques to sparsify a fine-tuned adapter model, enhancing the efficiency of PEFT.

Our proposed  $\mathbf{ID}^3$  method distinguishes itself from current selective PEFT methods by progressively selecting the parameters throughout fine-tuning steps, thereby capturing the change in parameter importance during training process. Additionally this will allow us to choose model checkpoints with incremental budgets, which is not possible in current methods. Furthermore,  $\mathbf{ID}^3$  leverages the magnitude and gradient of parameters, where the latter can be efficiently computed using any standard automatic differentiation tool (Baydin et al., 2018), thus avoiding extra computational delays.

## 3 Methodology

Motivated by the key challenges of the existing budget-driven selective PEFT methodologies, highlighted in Sections 1 and 2, we propose  $\mathbf{ID}^3$ , an iterative approach for calculating the parameter importance and incrementally selecting the top parameters for each training iteration. Figure 2 illustrates the workings of  $\mathbf{ID}^3$ . In this work, we introduce the terms *scalar parameter* and *tensor parameter*, where we refer to individual entries in the weight matrices as scalar parameters and the whole weight matrix as the tensor parameter.

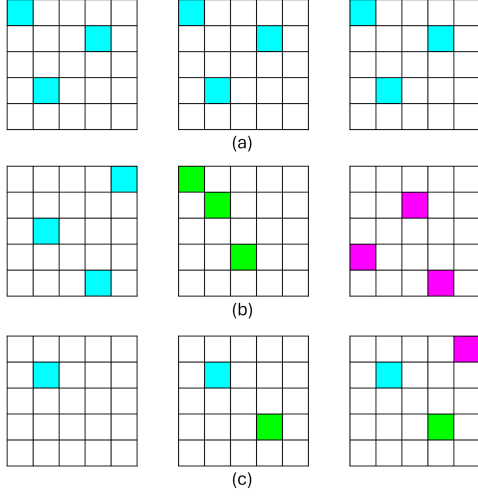


Figure 2:  $B$  (Budget) = 3,  $T$  (Total training steps) = 3 (**Top**): Static- $\mathcal{S}$  strategy where  $B$  number of parameters are chosen initially and used in all future training steps. (**Middle**): Repeat- $\mathcal{S}$  where  $B$  number of fresh parameters are chosen according to the heuristic at each training step. (**Bottom**): Increment- $\mathcal{S}$  where  $k$  ( $= B/T$ ) parameters are chosen at each training step according to the heuristic and added to the selected pool.

For instance, a tensor parameter in a BERT (Devlin et al., 2018) model can be the query matrix of an attention head. The query matrix has  $\frac{d^2}{n}$  scalar parameters where  $d$  is the hidden dimension, and  $n$  is the number of attention heads. We also formulate a common selective PEFT method as a heuristic function combined with a selection strategy. We identify three common selection strategies – (1) static- $\mathcal{S}$ , where the initial set of selected parameters according to the heuristic are reused throughout training, (2) repeat- $\mathcal{S}$ , where we use the heuristic repeatedly at each training step to find a (potentially) new selected set, and (3) increment- $\mathcal{S}$  where we accumulate the selected set over the training iterations. These selection strategies are illustrated in Figure 2. Existing selective PEFT methods use static- $\mathcal{S}$ , and  $\text{ID}^3$  uses increment- $\mathcal{S}$ . We provide results for repeat- $\mathcal{S}$  as baselines for comparison.

### 3.1 Determining Scalar Importance

Evaluating the scalar importance (*i.e.* importance of scalar parameters) of a neural network has always been a pivotal step in model pruning (Molchanov et al., 2019; Cheng et al., 2023). For a given neural model, parameterized with  $\theta$ , we calculate an importance function  $f : \mathbb{R}^2 \rightarrow [0, \infty]$  that measures a real-valued importance for

each parameter given its value  $\theta^i$  and the gradient,  $\nabla_{\theta^i}$ . Formally, we define the parameter importance function (also referred to as the heuristic function):

$$\mathcal{H}(\theta^i) = \frac{|\nabla_{\theta^i}|}{(|\theta^i| + \epsilon)^{exp}}, \quad (1)$$

where  $\epsilon \in (0, \infty]$  and  $exp \in [-\infty, \infty]$  are hyper-parameters to control the smoothing of the function and the effect of parameter magnitude on the final importance, respectively. The following theorem provides the mathematical justification behind the heuristic function.

**Definition 1.** Given the output distribution of  $y \sim p_\theta(\cdot|x)$ , where  $p_\theta(y|x) = f(x, y; \theta)$ , for a given input  $x$  and a model parameter  $\theta$ , the Fisher information matrix  $\mathcal{I}(\theta)$  is the variance  $\mathbb{E}_{x,y} \left[ \left( \frac{\partial}{\partial \theta} \log f(x, y; \theta) \right)^2 \middle| \theta \right] - \mathbb{E}_{x,y} \left[ \left( \frac{\partial}{\partial \theta} \log f(x, y; \theta) \right) \middle| \theta \right]^2$ .

Fisher information measures the amount of information the random variable  $x$  carries about the unknown model parameter  $\theta$  and is widely used to assess the model parameter importance.

**Theorem 1.** For  $\epsilon \geq 1$ ,  $\sqrt{\mathcal{I}(\theta)}$  is the upper bound of  $\mathbb{E}_{x,y} [\mathcal{H}(\theta)]$ .

**Proof of Theorem 1.** First, we show that  $\mathbb{E}_{x,y} \left[ \left( \frac{\partial}{\partial \theta} \log f(x, y; \theta) \right) \middle| \theta \right] = 0$ .

$$\begin{aligned} & \mathbb{E}_{x,y} \left[ \left( \frac{\partial}{\partial \theta} \log f(x, y; \theta) \right) \middle| \theta \right] \\ &= \int_x \int_y \frac{\frac{\partial}{\partial \theta} f(x, y; \theta)}{f(x, y; \theta)} f(x, y; \theta) dx \\ &= \frac{\partial}{\partial \theta} \int_x \int_y f(x, y; \theta) dx = \frac{\partial}{\partial \theta} \cdot 1 = 0 \end{aligned}$$

Therefore,

$$\begin{aligned} \mathcal{I}(\theta) &= \mathbb{E}_{x,y} \left[ \left( \frac{\partial}{\partial \theta} \log f(x, y; \theta) \right)^2 \middle| \theta \right] \\ \mathbb{E}_{x,y} [\mathcal{H}(\theta)] &= \frac{1}{(|\theta| + \epsilon)^{exp}} \mathbb{E}_{x,y} \left[ \left| \frac{\partial}{\partial \theta} \log f(x, y; \theta) \right| \right] \end{aligned}$$

Using Jensen’s inequality, we get,

$$\begin{aligned} \mathcal{I}(\theta) &= \mathbb{E}_{x,y} \left[ \left| \frac{\partial}{\partial \theta} \log f(x, y; \theta) \right|^2 \right] \\ &\geq \left( \mathbb{E}_{x,y} \left[ \left| \frac{\partial}{\partial \theta} \log f(x, y; \theta) \right| \right] \right)^2 \\ &= \left( \mathbb{E}_{x,y} [\mathcal{H}(\theta)] \right)^2 \cdot (|\theta| + \epsilon)^{2 \cdot exp} \end{aligned}$$

Hence, for  $\epsilon \geq 1$ ,  $\mathcal{I}(\theta) \geq \left( \mathbb{E}_{x,y} [\mathcal{H}(\theta)] \right)^2$ . Therefore, Theorem 1 justifies that parameters with maximum  $\mathcal{H}(\theta^i)$  have maximum Fisher importance.

---

### Algorithm 1 Incremental Parameter Update

---

**Require:** Unmasking scheduler  $\{u_t\}_{t=1}^T$ , number of training steps  $T$ , trainable model  $\theta_{(0)}$ , training dataset  $(X, Y)$ , learning rate  $\eta$

$t \leftarrow 0$   
 $\Lambda_0 \leftarrow \phi$   
**while**  $t < T$  **do**  
     $(x, y) \sim (X, Y)$  minibatch  
    Compute predicted output  $\hat{y} = p_{\theta_{(t)}}(\cdot|x)$   
    Compute loss  $l = \mathcal{L}(y, \hat{y})$   
    Compute gradient  $\nabla_{\theta_{(t)}} = \nabla_{\theta_{(t)}} l$   
    Compute parameter importance  $\mathcal{H}$  for parameters in  $\theta_{(t)} \setminus \Lambda_t$  using Equation 1  
    Find scalar parameters  $\lambda_t$  using Equation 2  
     $\Lambda_{t+1} \leftarrow \Lambda_t \cup \lambda_t$   
    Update parameter gradients  $\tilde{\nabla}_{\theta_{(t)}}$  using Equation 3  
    Perform parameter update  $\theta_{(t+1)} \leftarrow \theta_{(t)} + \eta \tilde{\nabla}_{\theta_{(t)}}$   
     $t \leftarrow t + 1$   
**end while**

---

### 3.2 Incremental Parameter Update

Suppose we want to fine-tune a pre-trained model parameterized by  $\theta_{(0)}$  (0 denotes the fine-tuning timestep), with  $|\theta_{(0)}| = N$  on a task for maximum  $T$  number of steps. Suppose we fix the budget of fine-tuning as  $B$ , i.e., we only fine-tune a maximum of  $B$  number of scalar parameters in the entire model training. The factor  $\frac{B}{N}$  is called *sparcity* of the model. We choose a suitable unmasking scheduler  $\{u_t\}_{t=1}^T$  that estimates the number of parameters to be updated in each iteration  $t$ . By default, we use a uniform scheduler where  $u_t = \frac{B}{T}$ . At the beginning of model fine-tuning, the unmasked parameters  $\Lambda_t = \phi$ . At each training iteration  $t$ , we measure the importance for each parameter in the set  $\theta_{(t-1)} \setminus \Lambda_{t-1}$  using Equation 1 and determine the incremental unmasked parameters  $\Lambda_t$  such that

$$\max_{\lambda_t} \min_{\theta^i \in \lambda_t} \{\mathcal{H}(\theta^i)\} \text{ s.t. } |\lambda_t| = u_t \quad (2)$$

Finally, the set of unmasked parameters is updated as  $\Lambda_t = \Lambda_{t-1} \cup \lambda_t$ . During the forward pass, we compute the task-specific loss  $\mathcal{L}(y, p_{\theta_{(t)}}(y|x))$ . The gradients  $\nabla_{\theta_{(t)}}$  are set to

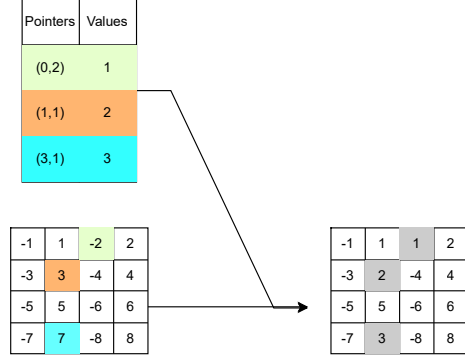


Figure 3: **Bottom left:** A tensor parameter in the base pre-trained model, **Top left:** Table of pointers and values of the updated scalar parameters after back-propagation, **Right:** Updated model parameter after back-propagation.

zeros, for parameters not in the unmask set  $\Lambda_t$  to obtain  $\tilde{\nabla}_{\theta_t}$ . Formally,

$$\tilde{\nabla}_{\theta_t^i} = \begin{cases} \nabla_{\theta_t^i}, & \text{if } \theta_t^i \in \Lambda_t \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

Finally, the parameters are updated with back-propagation using the updated gradients  $\tilde{\nabla}_{\theta_{(t)}}$ . Algorithm 1 formalizes the **ID**<sup>3</sup> incremental parameter update algorithm.

With the incremental parameter selection and update, the total number of parameter updates can be calculated as

$$\mathcal{U}_{dynamic} = \sum_{t=0}^{T-1} \sum_{i=0}^t u_t$$

For uniform unmasking scheduler,

$$\mathcal{U}_{dynamic} = \sum_{t=0}^{T-1} \sum_{i=0}^t \frac{B}{T} = \frac{T+1}{2} B$$

For static-masking-based PEFT techniques, the total number of parameter updates is

$$\mathcal{U}_{static} = \sum_{t=0}^T B = T \cdot B$$

Hence,  $\mathcal{U}_{dynamic} = \frac{\mathcal{U}_{static}}{2}$  (when  $T \gg 1$ ). Therefore, the incremental selection can reduce the effective number of gradient updates by a factor of 2.

### 3.3 Efficient Processing of Sparse Masks

Storing and loading the sparse masks requires efficient handling of the masked scalar parameters.

For storing the sparse weights, we store only the weights of the unmasked scalar parameters and their corresponding pointers. Since the maximum dimension of any tensor does not typically exceed 2, we need to store at most two indices for any given scalar parameter. These indices can be stored using a 32-bit unsigned integer. Each updated model parameter can be stored using 64-bit double floating point numbers. Therefore, we can reduce the space complexity of the masks to  $\mathcal{O}(2 \times 32 \times B + 64 \times B) = \mathcal{O}(B)$ . While loading, we can use these pointers (stored in the form of tensors) to index into the tensor parameters and replace the pretrained ones with the stored ones which were learned during selective fine-tuning. Figure 3 summarizes the process of handling sparse masks.

## 4 Experimental Setup

### 4.1 Datasets and Tasks

To evaluate the effectiveness of our proposed method, we conduct exhaustive experiments across three distinct tasks: text classification, token classification, and text generation.

For text classification, we use eight tasks from the GLUE benchmark (Wang et al., 2018): CoLA, MRPC, RTE, STS-B, SST-2, MNLI-m/mm, QNLI, and QQP. In line with previous studies (Liao et al., 2023; Sung et al., 2021; Zaken et al., 2021), we exclude the WNLI task due to its poor performance with pre-trained language models. On token classification, we experiment with the named entity recognition (NER) task using the CoNLL-2003 dataset (Tjong Kim Sang and De Meulder, 2003). For these nine tasks, we fine-tune the model using the training splits and evaluate its performance on the validation splits.

We consider six generative arithmetic reasoning tasks for text generation: GSM8K, SVAMP, MultiArith, AddSub, AQuA, and SingleEq. We fine-tune our models on the Math10K dataset, as curated by Hu et al. (2023), and evaluate them on the test splits of the datasets above. Detailed descriptions of these datasets and tasks are provided in Section 8.1 and Table 6 of Appendix.

### 4.2 Models

For NLU and NER tasks, we use the pre-trained encoder-only DeBERTa-v3-base (He et al., 2021b) and RoBERTa-base (Liu et al., 2019) models as the backbone. For generative tasks, we use

the pre-trained Llama-7b (Touvron et al., 2023) model. All the pre-trained model weights are obtained from Huggingface (Wolf et al., 2020).

### 4.3 Toolkit Implementation

A significant contribution of our work is the implementation of the *selective-peft-toolkit*<sup>2</sup>. We use PyTorch (Paszke et al., 2019) and the Huggingface Transformers library (Wolf et al., 2020) for implementing the toolkit. We implement the following selective PEFT baselines in our toolkit: (1) BitFit (Zaken et al., 2021) involves fine-tuning only the bias terms in a pre-trained model; (2) PaFi (Liao et al., 2023) selects the pre-trained parameters with the smallest magnitude and trains only these parameters during fine-tuning; (3) **ID**<sup>3</sup>.

The toolkit allows integration of these selective PEFT methods into the original pre-trained models as well as into any additional neural modules such as Adapters (Houlsby et al., 2019; Pfeiffer et al., 2020) and LoRA (Hu et al., 2021). We also provide methods for storing and loading the sparse weights memory-efficiently, enabling end-to-end training and evaluation workflows.

**Hyperparameters.** Batch size, learning rates and other PEFT method-specific hyper-parameters are provided in Section 8.2 and Table 7a and Table 7b of Appendix. All the models are trained on Nvidia A100 and A6000 GPUs.

## 5 Results

### 5.1 Text Classification

We report the results on GLUE tasks in Table 1. **ID**<sup>3</sup> achieves an average score of 89.15% with a budget of 320K, surpassing the best-performing baseline (Fish) by over 1% and even outperforming the FFT baseline (88.86%). A similar comparison holds at smaller budget levels, with **ID**<sup>3</sup> outperforming other selective baselines by >1%.

We further evaluate the effectiveness of **ID**<sup>3</sup> with other adapters integrated with pre-trained language models. Table 2 reports the performance of the DeBERTa-v3 model with rank 8 (indicated by r=8) LoRA adapter, with and without **ID**<sup>3</sup>. With a budget of 320K (sparsity 76%), **ID**<sup>3</sup> outperforms full LoRA fine-tuning by a margin of 0.17%. Interestingly, LoRA sparsified with both **ID**<sup>3</sup> and PaFi beats the dense LoRA model on

<sup>2</sup><https://github.com/Aradhya2002/selective-peft-toolkit>

Budget	Method	MNLI-m	MNLI-mm	QQP	QNLI	SST-2	STS-B	CoLA	MRPC	RTE	Avg
184M	Full-FT	90.34 <sub>0.18</sub>	90.51 <sub>0.19</sub>	92.11 <sub>0.28</sub>	94.24 <sub>0.10</sub>	96.33 <sub>0.11</sub>	91.04 <sub>0.48</sub>	71.43 <sub>0.72</sub>	89.95 <sub>1.07</sub>	83.75 <sub>1.81</sub>	88.86
103K	R-Mask	85.18 <sub>0.43</sub>	85.87 <sub>0.04</sub>	87.09 <sub>0.61</sub>	89.31 <sub>2.04</sub>	93.61 <sub>0.53</sub>	84.05 <sub>3.91</sub>	61.14 <sub>3.51</sub>	78.59 <sub>4.42</sub>	60.53 <sub>6.68</sub>	80.60
	Fish	88.14 <sub>0.18</sub>	88.36 <sub>0.16</sub>	87.50 <sub>0.06</sub>	92.29 <sub>0.24</sub>	95.22 <sub>0.27</sub>	91.68 <sub>0.31</sub>	69.16 <sub>0.76</sub>	<u>90.52</u> <sub>0.51</sub>	<u>84.72</u> <sub>0.75</sub>	86.95
	PaFi	88.21 <sub>0.15</sub>	88.29 <sub>0.19</sub>	89.30 <sub>0.35</sub>	93.63 <sub>0.31</sub>	<b>95.84</b> <sub>0.13</sub>	89.97 <sub>0.70</sub>	68.45 <sub>1.09</sub>	<u>90.20</u> <sub>0.65</sub>	81.83 <sub>1.26</sub>	86.97
	BitFit	88.64 <sub>0.10</sub>	88.39 <sub>0.18</sub>	88.94 <sub>0.26</sub>	93.26 <sub>0.22</sub>	95.38 <sub>0.13</sub>	89.80 <sub>1.12</sub>	69.08 <sub>1.07</sub>	89.54 <sub>1.35</sub>	80.87 <sub>2.60</sub>	86.43
	<b>ID<sup>3</sup></b>	<b>89.38</b> <sub>0.07</sub>	<b>89.64</b> <sub>0.10</sub>	<b>89.97</b> <sub>0.01</sub>	<b>93.67</b> <sub>0.17</sub>	95.64 <sub>0.23</sub>	<u>92.10</u> <sub>0.13</sub>	<b>70.62</b> <sub>0.44</sub>	<u>91.01</u> <sub>0.14</sub>	<u>87.12</u> <sub>2.17</sub>	<b>88.13</b>
320K	R-Mask	88.30 <sub>0.39</sub>	88.33 <sub>0.17</sub>	89.07 <sub>0.47</sub>	92.30 <sub>0.54</sub>	95.30 <sub>0.23</sub>	88.26 <sub>0.66</sub>	66.19 <sub>0.98</sub>	86.11 <sub>0.75</sub>	74.73 <sub>2.53</sub>	85.40
	Fish	88.99 <sub>0.05</sub>	89.70 <sub>0.08</sub>	88.92 <sub>0.28</sub>	93.99 <sub>0.06</sub>	95.60 <sub>0.06</sub>	<b>92.09</b> <sub>0.13</sub>	69.79 <sub>1.04</sub>	<u>90.69</u> <sub>0.49</sub>	<b>87.85</b> <sub>0.55</sub>	88.07
	PaFi	89.61 <sub>0.10</sub>	89.73 <sub>0.15</sub>	90.35 <sub>0.22</sub>	93.93 <sub>0.11</sub>	<b>96.14</b> <sub>0.07</sub>	90.37 <sub>0.24</sub>	68.91 <sub>1.16</sub>	<u>90.52</u> <sub>0.62</sub>	81.59 <sub>2.01</sub>	87.68
	<b>ID<sup>3</sup></b>	<b>89.73</b> <sub>0.07</sub>	<b>89.91</b> <sub>0.09</sub>	<b>90.42</b> <sub>0.20</sub>	<b>94.06</b> <sub>0.10</sub>	95.95 <sub>0.07</sub>	<u>92.07</u> <sub>0.22</sub>	<b>72.07</b> <sub>0.99</sub>	<b>91.26</b> <sub>0.28</sub>	<u>87.73</u> <sub>1.45</sub>	<b>89.15</b>

Table 1: Performance comparison of different selective PEFT methods on GLUE tasks with DeBERTa-v3 (He et al., 2021b) pretrained model. For each experiment, we report the mean and standard deviation obtained across top three of four different runs. We highlight the best PEFT baseline within a comparable budget in **bold**. We underline the tasks where the PEFT techniques outperform the FFT counterpart. DeBERTa-v3 has only 103K bias terms; therefore, BitFit is applicable only with 103K budget. R-mask refers the baseline with a random static mask.

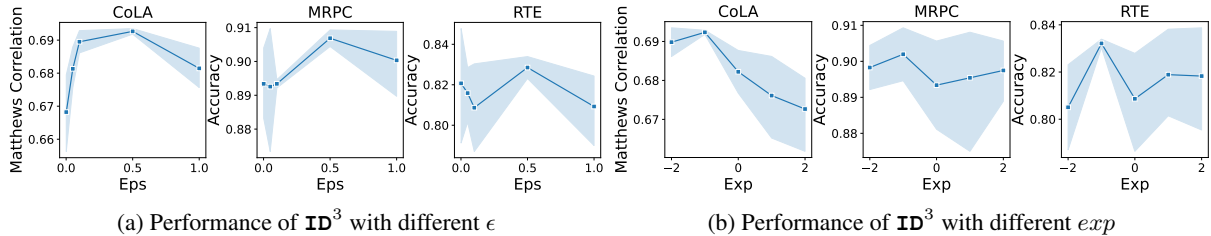


Figure 4: Performance of **ID<sup>3</sup>** under different  $\epsilon$  and  $exp$  values with DeBERTa-V3 backbone model.

four of nine GLUE tasks, indicating the importance of sparsification of adapters for more efficient and effective fine-tuning. An empirical study with adapters (Pfeiffer et al., 2020) narrates a similar story as shown in Table 3. With a budget of only 8M (sparsity 96%), **ID<sup>3</sup>** can improve the performance of an adapter-integrated RoBERTa-base by a margin of 1.09%. SparseAdapter, another popular sparsification technique for adapters, falls short by 0.91% than **ID<sup>3</sup>**.

We further evaluate **ID<sup>3</sup>** with different  $\epsilon$  and  $exp$  values and report the performance across three different NLU tasks. Figure 4a highlights an interesting trend where the best performance is achieved typically with  $\epsilon = 0.5$ . Low  $\epsilon$  values have less smoothing effect and, therefore, prevent parameters with low magnitude from being unmasked unfairly. The optimal selection remains close at 0.5, which has more balanced effect on all the parameters. A consistent trend is also observed with  $exp$  (c.f. Figure 4b), where  $exp = -1$  consistently performs better than other values. This indicates that parameter magnitude has a positive dependence on the parameter importance.

## 5.2 Token Classification

We present the CoNLL benchmark results in Table 4. FFT, with 184M parameters, sets the baseline with an F1 score of 96.69% and a standard deviation of 0.18. Among PEFT methods, **ID<sup>3</sup>** excels with 103K parameters, achieving a top F1 score of 95.86% and a low standard deviation of 0.07, indicating both high performance and consistency. Fish follows with an F1 score of 95.43% (SD=0.27) and PaFi scores 95.23% (SD=0.20). BitFit scores 94.62% but has a higher standard deviation of 0.84, suggesting greater variability. With a 320K parameter budget, **ID<sup>3</sup>** improves to an F1 score of 96.17 (SD=0.14), surpassing other methods. PaFi scores 96.04 (SD=0.29) and Fish scores 95.99 (SD=0.14). Notably, **ID<sup>3</sup>** outperforms Fish by 0.43% and PaFi by 0.63% at 103K parameters, demonstrating robustness. At 320K parameters, **ID<sup>3</sup>**'s performance approaches that of the FFT method, proving it to be a highly effective alternative.

## 5.3 Text Generation

We report the results for the mathematical reasoning tasks in Table 5. Llama-7B, fine-tuned with LoRA ( $r=32$ ), achieves an average score of 59.5%,

Budget	Method	MNLI-m	MNLI-mm	QQP	QNLI	SST-2	STS-B	CoLA	MRPC	RTE	Avg
1.33M	LoRA (r=8)	90.47 <sub>0.23</sub>	90.46 <sub>0.12</sub>	91.95 <sub>0.12</sub>	93.76 <sub>0.36</sub>	95.57 <sub>0.21</sub>	91.86 <sub>0.29</sub>	69.73 <sub>1.42</sub>	89.71 <sub>1.32</sub>	85.32 <sub>0.86</sub>	88.76
320K	PaFi + LoRA (r=8)	90.12 <sub>0.1</sub>	90.06 <sub>0.42</sub>	<b>91.61</b> <sub>0.34</sub>	<b>94.31</b> <sub>0.12</sub>	<b>96.03</b> <sub>0.06</sub>	91.19 <sub>0.18</sub>	<b>70.5</b> <sub>0.59</sub>	<b>90.28</b> <sub>0.57</sub>	84.72 <sub>0.55</sub>	88.76
320K	<b>ID</b> <sup>3</sup> + LoRA (r=8)	<b>90.26</b> <sub>0.25</sub>	<b>90.18</b> <sub>0.21</sub>	91.48 <sub>0.26</sub>	<b>94.17</b> <sub>0.07</sub>	<b>95.76</b> <sub>0.12</sub>	<b>91.65</b> <sub>0.21</sub>	69.07 <sub>0.55</sub>	<b>90.69</b> <sub>0.25</sub>	<b>87.12</b> <sub>0.21</sub>	<b>88.93</b>

Table 2: Performance of PaFi and **ID**<sup>3</sup> with LoRA+pretrained DeBERTa-v3 model on GLUE tasks.

Budget	Method	STS-B	CoLA	MRPC	RTE	Average
201M	Pfeiffer	90.78	59.05	89.21	76.53	78.89
8M	SparseAdapter + Pfeiffer	<b>90.88</b>	58.95	89.41	<b>77.03</b>	<b>79.07</b>
8M	<b>ID</b> <sup>3</sup> + Pfeiffer	90.71	<b>59.84</b>	<b>89.95</b>	<b>79.42</b>	<b>79.98</b>

Table 3: Performance of **ID**<sup>3</sup> compared with SparseAdapter (He et al., 2022) on Pfeiffer adapter (Pfeiffer et al., 2020) on pretrained RoBERTa-base (Liu et al., 2019) model.

Budget	Full-FT	Fish	PaFi	BitFit	<b>ID</b> <sup>3</sup>	R-Mask
103K	-	95.43 <sub>0.27</sub>	95.23 <sub>0.2</sub>	94.62 <sub>0.84</sub>	<b>95.86</b> <sub>0.07</sub>	85.42 <sub>7.03</sub>
320K	-	95.99 <sub>0.12</sub>	96.04 <sub>0.29</sub>	-	<b>96.17</b> <sub>0.14</sub>	93.74 <sub>1.32</sub>
184M	96.69 <sub>0.18</sub>	-	-	-	-	-

Table 4: Performance of selective fine-tuning methods with DeBERTa-v3 model on NER task with different budgets. As the DeBERTa model has total 103K bias terms, BitFit is applicable only with 103K budget.

demonstrating strong performance across all tasks, particularly in MultiArith (95.5%) and SingleEq (81.7%). This serves as the baseline for comparison with various PEFT methods. At a reduced budget of 3.5M parameters, LoRA (r=2) maintains robust performance with an average score of 58.1%, showing notable performance in MultiArith (96.7%) but a slight decline across other tasks compared to the 201M budget. **ID**<sup>3</sup> with LoRA (r=32) achieves an average score of 58.6%, slightly outperforming LoRA (r=2) (which has an equivalent budget), with notable scores including 80.7% for AddSub and 79.3% for SingleEq, indicating that **ID**<sup>3</sup> improve performance by sparsifying larger rank LoRA modules. PaFi with LoRA shows an average score of 57.0%, with its best performance in MultiArith (92.3%), but falls behind **ID**<sup>3</sup> and LoRA (r=32) full fine-tuning in most tasks.

Increasing the LoRA rank to 4 (budget 7M) improves its average score to 58.5%, with significant gains in GSM8K (36.9%) and MultiArith (94.8%). **ID**<sup>3</sup> with a 7M budget achieves the highest average score in this category at 59.1%, showing strong performance in GSM8K (38.3%) and SVAMP (47.2%), suggesting that **ID**<sup>3</sup> effectively utilizes the additional parameters. PaFi, with an average score of 56.4%, shows consistent results but lags behind other methods in Multi-

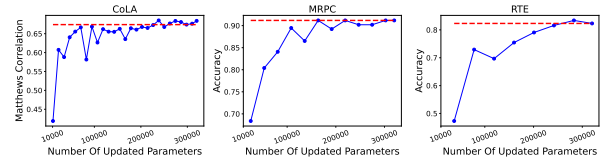


Figure 5: Dev performance at different model checkpointing with **ID**<sup>3</sup>. The red line highlights the performance obtained with repeat- $\mathcal{S}$  parameter selection.

Arith (92.3%) and SVAMP (42.9%). **ID**<sup>3</sup>, along with higher rank LoRA, consistently performs well across different parameter budgets, specifically excelling in the 7M budget category with the highest average score of 59.1%. Notably, **ID**<sup>3</sup> surpasses the performance of the base LoRA in the same budget range, demonstrating its efficiency and effectiveness.

## 6 Discussions

We further study the behaviors of fine-tuned models under the incremental masking and answer some of the questions relevant to selective fine-tuning.

**Why do we need incremental parameter selection?** We explore a variant of **ID**<sup>3</sup> with repeat- $\mathcal{S}$  selection strategy instead of increment- $\mathcal{S}$ , which we call repeat-**D**<sup>3</sup>. Figure 5 highlights that for all the tasks – CoLA, MRPC and RTE, incremental selection with  $< B$  number of updated parameters can exhibit better validation performance than the repeat- $\mathcal{S}$  selection method where  $B$  number of parameters are unmasked and fine-tuned in each training step. As highlighted in Section 1, a completely exploration-based selection strategy can fine-tune too many redundant parameters, potentially leading to worse out-of-distribution performance. To understand how the maximum budget impacts the fine-tuning process, we experiment with different budgets, as highlighted in Figure 6. For budgets as low as 80K (sparsity 99.95%), repeat- $\mathcal{S}$  selection might work better than increment- $\mathcal{S}$  (the selection strategy of **ID**<sup>3</sup>) selection. This could be due to significantly fewer trainable parameters  $\frac{B}{S}$  during the initial



Budget	Method	AddSub	MultiArith	SingleEq	GSM8K	AQuA	SVAMP	Avg.
201M	LoRA (r=32)	81.3	95.5	81.7	34.1	17.7	46.7	59.5
3.5M	LoRA (r=2)	78.2	<b>96.7</b>	76.6	<b>35.3</b>	<b>16.9</b>	44.9	58.1
3.5M	PaFi + LoRA (r=32)	78.7	92.3	76.8	33.9	<b>16.9</b>	43.2	57.0
3.5M	<b>ID<sup>3</sup></b> + LoRA (r=32)	<b>80.7</b>	95.8	<b>79.3</b>	<u>34.3</u>	15.7	<b>45.7</b>	<b>58.6</b>
7M	LoRA (r=4)	79.2	<b>94.8</b>	77.9	<u>36.9</u>	<b>17.3</b>	45.0	58.5
7M	PaFi + LoRA (r=32)	75.7	92.3	76.2	<u>34.9</u>	16.5	42.9	56.4
7M	<b>ID<sup>3</sup></b> + LoRA (r=32)	<b>79.5</b>	<b>94.8</b>	<b>79.7</b>	<b>38.3</b>	15.0	<b>47.2</b>	<b>59.1</b>

Table 5: Results on mathematical reasoning tasks with Llama-7B backbone model with LoRA.

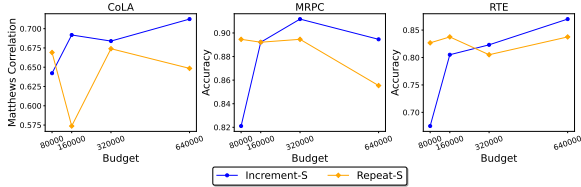


Figure 6: Performance of  $ID^3$  under with incremental masking and repeat- $S$  masking with different budgets.

fine-tuning stage. However, increment- $S$  selection performs substantially better than repeat- $S$  selection for higher budgets. Interestingly, for higher budgets (sparsity  $< 99.82\%$ ), the performance for repeat- $S$  selection drops significantly. These analyses also highlight the importance of dynamic parameter and budget selection. Traditional selective PEFT methods use a fixed budget throughout the model fine-tuning, which may not be optimal for a given task. On the other hand, using incremental budgeting like  $ID^3$ , we can save the intermediate checkpoints with only incremental parameters being updated. Therefore, by preventing unnecessary parameter update,  $ID^3$  can converge faster and lead to better out-of-distribution performance.

### Can we sparsify tensor parameters using $ID^3$ ?

A relevant research objective with selective PEFT techniques is to study their ability to sparsify tensor parameters. Significant masking memory can be optimized if a selective method unmasks scalar parameters from a few selective tensor parameters. For a model with  $M$  number of tensor parameters  $\{P^i\}_{i=1}^M$  fine-tuned with  $T$  steps, we define ‘tensor sparsity’ as the number of parameters  $P^j$  such that  $\nexists \theta^k \in P^j \cap \Lambda_T$ . Figure 8 highlights the tensor sparsity for  $ID^3$  with increment- $S$  and repeat- $S$  selection at different training iterations. For all three tasks – CoLA, MRPC and RTE, tensor sparsity remains close to one for  $ID^3$  at the beginning. As the training continues, the tensor sparsity reduces as more scalar parameters are explored.

However, the reduction in tensor sparsity stabilizes after a few training steps, indicating more exploration from the same tensor parameters. A similar behavior is also observed with repeat- $S$  parameter selection. However, with this approach, the tensor sparsity remains critically low, as this selection method exceeds the budget and potentially fine-tunes the entire model.

### Are all the tensor parameters equally important in selective fine-tuning?

To answer this question, we compute the sparsity probability for each tensor parameter  $P_j$  as  $\frac{|P_j \cap \Lambda_T|}{|P_j|}$ . Using this probability distribution over all the tensor parameters, we calculate the sparsity entropy of the fine-tuned model. A high entropy indicates uniform sparsity probability across different parameters, indicating uniform parameter importance. Figure 9 suggests that for increment- $S$  selection, initially, the entropy increases, indicating more exploration of important scalar parameters from different tensor parameters. However, after a few training iterations, the model performs more exploitation by selecting scalar parameters from the same tensor parameters. On the other hand, a repeat- $S$  parameter selection strategy performs exploration from the beginning, assigning importance to most of the tensor parameters. Figure 7 highlights the scalar parameter allocation for different tensor parameters for the DeBERTa-v3 model on the CoLA task. With both increment- $S$  and repeat- $S$  selection strategies, the initial parameter allocation remains sparse. After training, the sparsity within the tensor parameter reduces. However, it is interesting that increment- $S$  selection introduces more tensor sparsity than repeat- $S$  selection. These behaviors justify that increment- $S$  selection is essential for assessing tensor parameter importance and sparse updation of model parameters.

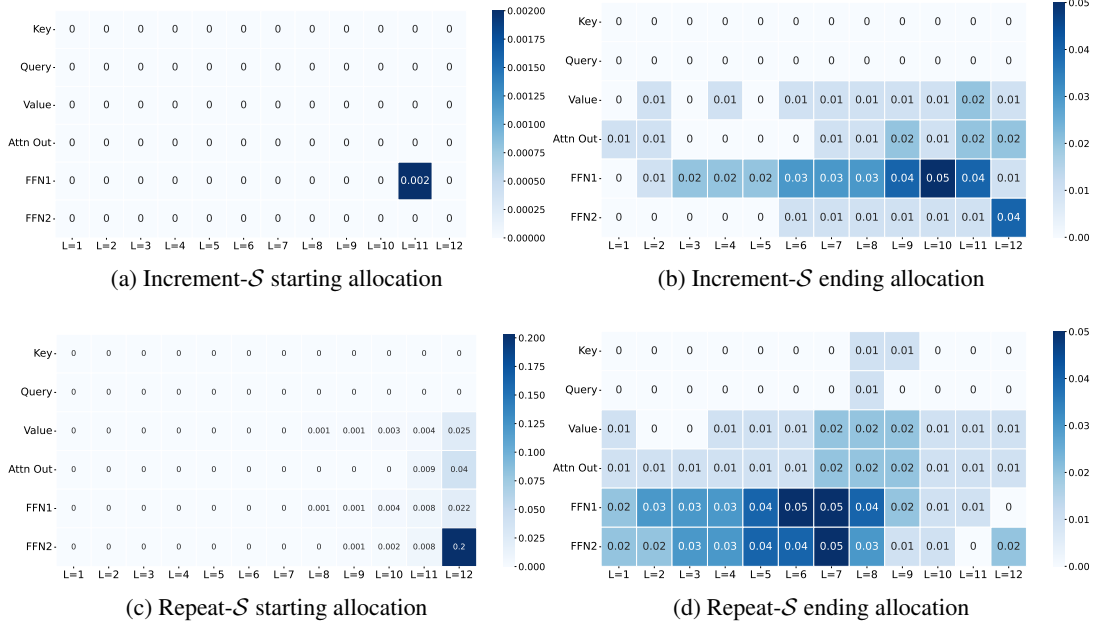


Figure 7: Allotted budget to different model parameters under increment- $\mathcal{S}$  and repeat- $\mathcal{S}$  masking for CoLA task.

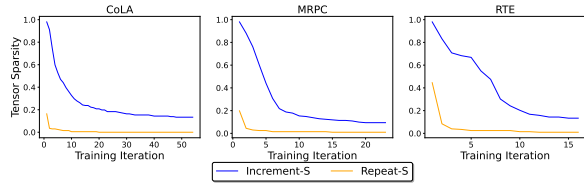


Figure 8: Analysis of tensor sparsity with increment- $\mathcal{S}$  and repeat- $\mathcal{S}$  selection strategies.

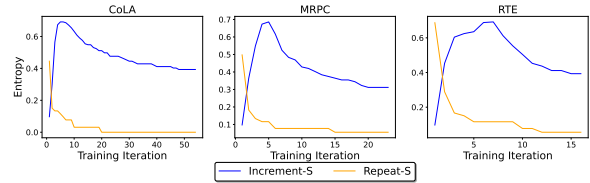


Figure 9: Sparsity entropy with increment- $\mathcal{S}$  and repeat- $\mathcal{S}$  selection strategies.

## 7 Conclusion

In this paper, we introduced  $\mathbf{ID}^3$ , a novel parameter-efficient fine-tuning (PEFT) technique using incremental-masking-based parameter selection to enhance the fine-tuning of large language models.  $\mathbf{ID}^3$  dynamically evaluates and updates parameter importance, effectively balancing exploration and exploitation. Our extensive evaluations showed that  $\mathbf{ID}^3$  significantly outperforms traditional PEFT methods, with significantly less number of gradient updates. Additionally,  $\mathbf{ID}^3$  integrates seamlessly with other PEFT methodologies, showcasing its versatility. We provide an open-source toolkit with four selective PEFT techniques to support reproducibility and further research. This study marks a significant advancement in PEFT, improving performance while reducing computational overhead and enabling broader scalability of LLMs.

**Limitations and Future Scope.** While selec-

tive PEFT methods do reduce the number of gradient updates (with  $\mathbf{ID}^3$  achieving competitive performance in half as many updates), the current implementation does not fully leverage this efficiency due to limitations in low-level C++ libraries, which predominantly support dense updates. To overcome this, future work will aim to integrate our method directly into the PyTorch library at a lower level, which could better realize the theoretical speedup discussed. Additionally, an intriguing research direction is to explore the activation of parameters updated through selective PEFT from a mechanistic perspective. Our current work provides some insights into this area, but a more detailed understanding could further illuminate how selective fine-tuning affects large pre-trained language models and enhance explainability in this field.

## References

- Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. 2018. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18(153):1–43.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.
- Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. [SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation](#). In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada. Association for Computational Linguistics.
- Hongrong Cheng, Miao Zhang, and Javen Qin-feng Shi. 2023. A survey on deep neural network pruning-taxonomy, comparison, analysis, and recommendations. *arXiv preprint arXiv:2308.06767*.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- William B. Dolan and Chris Brockett. 2005. [Automatically constructing a corpus of sentential paraphrases](#). In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.
- Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. [The third PASCAL recognizing textual entailment challenge](#). In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 1–9, Prague. Association for Computational Linguistics.
- Demi Guo, Alexander M Rush, and Yoon Kim. 2020. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463*.
- Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021a. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.
- Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021b. [Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing](#).
- Shwai He, Liang Ding, Daize Dong, Miao Zhang, and Dacheng Tao. 2022. Sparseadapter: An easy approach for improving the parameter-efficiency of adapters. *arXiv preprint arXiv:2210.04284*.
- Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–533.
- Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.
- Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. 2023. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*.

- Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597.
- Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.
- Baohao Liao, Yan Meng, and Christof Monz. 2023. Parameter-efficient fine-tuning without introducing new latency. *arXiv preprint arXiv:2305.16742*.
- Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 158–167.
- Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965.
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.
- Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11264–11272.
- Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. **Pytorch: An imperative style, high-performance deep learning library**. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.
- Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.
- Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*.
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.
- Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. **SQuAD: 100,000+ questions for machine comprehension of text**. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.
- Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1743–1752.
- Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. **Recursive deep models for semantic compositionality over a sentiment treebank**. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.
- Yi-Lin Sung, Varun Nair, and Colin A Raffel. 2021. Training neural networks with fixed sparse masks. *Advances in Neural Information Processing Systems*, 34:24193–24205.

- Erik F. Tjong Kim Sang and Fien De Meulder. 2003. [Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition](#). In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.
- Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.
- Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.
- Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.
- Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. [A broad-coverage challenge corpus for sentence understanding through inference](#). In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.
- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. [Transformers: State-of-the-art natural language processing](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.
- Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.
- Feiyu Zhang, Liangzhi Li, Junhao Chen, Zhouqiang Jiang, Bowen Wang, and Yiming Qian. 2023a. Increlora: Incremental parameter allocation method for parameter-efficient fine-tuning. *arXiv preprint arXiv:2308.12043*.
- Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023b. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*.
- Yaoming Zhu, Jiangtao Feng, Chengqi Zhao, Mingxuan Wang, and Lei Li. 2021. Counter-interference adapter for multilingual machine translation. *arXiv preprint arXiv:2104.08154*.

## 8 Appendix

### 8.1 Datasets and tasks

We evaluate all the methods using the eight tasks from the GLUE benchmark as below.

1. RTE (Recognizing Textual Entailment) ([Giampiccolo et al., 2007](#)) - Each example consists of two sentences, and the task is to predict whether the second sentence entails the first sentence.
2. MRPC (Microsoft Research Paraphrase Corpus) ([Dolan and Brockett, 2005](#)) - Predict whether the given two sentences are semantically equivalent.
3. CoLA (Corpus of Linguistic Acceptability) ([Warstadt et al., 2019](#)) - The task is to predict whether the given sentence is linguistically acceptable.
4. STS-B (Semantic Textual Similarity Benchmark) ([Cer et al., 2017](#)) - The task is to predict how similar the given two sentences are on a scale of 1 to 5.
5. SST-2 (Stanford Sentiment Treebank) ([Socher et al., 2013](#)) - The task is to predict whether the sentiment of a given movie review is positive or negative.

6. QNLI (Question-answering NLI) (Rajpurkar et al., 2016)- Each example consists of a question and a context. The task is to predict whether the given context contains the answer to the question.
7. QQP (Quora Question Pairs) (Wang et al., 2018) - The task is to determine whether a given question pair is semantically equivalent.
8. MNLI (Multi-Genre Natural Language Inference) (Williams et al., 2018) - Given a premise sentence and a hypothesis sentence, the task is to predict whether the premise entails the hypothesis or contradicts the hypothesis or neither. This data-set has two validation sets - matched (in-domain) and mismatched (cross-domain) data.

For token classification, we use the shared task of CoNLL2003 (Tjong Kim Sang and De Meulder, 2003) that focuses on language-independent named-entity recognition. The goal is to classify each token into four entities - persons, locations, organizations and miscellaneous entities that do not belong to the previous three groups.

We use six datasets from the Math Reasoning benchmark for text generation tasks.

1. GSM8K (Cobbe et al., 2021) - This dataset contains diverse grade school math word problems. The task is to perform a sequence of elementary calculations to get final answer.
2. SVAMP (Patel et al., 2021) - This dataset is developed by applying simple variations to one-unknown arithmetic word problems of grade level up to 4.
3. MultiArith (Roy and Roth, 2015) - This dataset contains multi-step arithmetic word problems that use the basic operations. eg. addition followed by subtraction, subtraction followed by division etc.
4. AddSub (Hosseini et al., 2014) - This corpus contains arithmetic problems with addition and subtraction.
5. AQuA (Ling et al., 2017) - This dataset contains algebraic word problems along with answer rationales.

Domain	Dataset	# train	# validation	# test
GLUE	RTE	2.5K	277	3K
	MRPC	3.7K	408	1.7K
	CoLA	8.5K	1K	1K
	STS-b	5.7K	1.5K	1.4K
	SST-2	67K	872	1.8K
	QNLI	105K	5.5K	5.5K
	QQP	364K	40K	390K
	MNLI	393K	m - 10K mm - 10K	10K 10K
NER	CoNLL2003	14K	3.2K	3.5K
Math Reasoning	Math10k	10K	-	-
	GSM8K	8.8K	-	1319
	SVAMP	-	-	1000
	MultiArith	-	-	600
	AddSub	-	-	395
	AQuA	100K	-	254
	SingleEq	-	-	508

Table 6: Datasets and data splits for different tasks used in the paper.

6. SingleEq (Koncel-Kedziorski et al., 2015) - This dataset contains sentences expressing mathematical relations that form a single equation.

The train, validation and test splits of all the datasets are shown in Table 6.

## 8.2 Hyperparameters

For NLU and NER tasks, we use batch size 16 across all models and PEFT methods. We choose four learning rates:  $\{1 \times 10^{-4}, 3 \times 10^{-4}, 5 \times 10^{-4}, 7 \times 10^{-4}\}$ <sup>3</sup> to fine-tune the models. These learning rates were chosen without bias towards a particular method and considering the common pattern of choosing learning rates in the  $1 \times 10^{-3}$  to  $1 \times 10^{-4}$ . We average the best three out of these four runs for statistical reliability. We use a batch size of 4 for generative tasks and fine-tune the models with a learning rate of  $3 \times 10^{-4}$  for 3 epochs. These and other PEFT method-specific hyper-parameters are shown in Table 7a.

Metric used for evaluation, number of epochs, number of eval steps and max sequence length are shown in Table 7b.

<sup>3</sup>For full fine-tuning these learning-rates are very high and the model diverges. Hence for FFT we choose  $\{5 \times 10^{-6}, 7 \times 10^{-6}, 1 \times 10^{-5}, 3 \times 10^{-5}\}$ .

	Category	NLU	NER	Generative Tasks	
PEFT Method	Hyper-parameter	All tasks	Conll2003	All tasks	
All methods	batch size	16	16	4	
		$1 \times 10^{-4}$	$1 \times 10^{-4}$		
	learning rate	$3 \times 10^{-4}$	$3 \times 10^{-4}$	$3 \times 10^{-4}$	
		$5 \times 10^{-4}$	$5 \times 10^{-4}$		
seed	$7 \times 10^{-4}$	$7 \times 10^{-4}$			
		{6, 7, 8, 9}	{6, 7, 8, 9}	42	
ID <sup>3</sup>	<i>exp</i>	2	2	0	
	$\epsilon$	1	1	1	
Fish	num_samples	1024	1024		
	sample_type	"label"	"label"	-	
	grad_type	"square"	"square"		
LoRA	lora_r	8		8	
	lora_alpha	8		16	
	lora_modules	query_proj			query_proj
		key_proj		-	key_proj
		value_proj			value_proj
		attention.output.dense			up_proj
		intermediate.dense			down_proj
output.dense					

(a) Common and PEFT method specific hyper-parameters

Benchmark	Dataset	Metric	Epochs	Eval_Steps	Max Seq Length
GLUE	RTE	Accuracy	30	100	256
	MRPC	Accuracy	30	100	256
	CoLA	Matthews Correlation	20	200	256
	STS-B	Avg of Spearman and Pearson Corr.	15	200	256
	SST-2	Accuracy	7	500	256
	QNLI	Accuracy	7	1000	256
	QQP	Accuracy	3	4000	256
	MNLI	Accuracy	3	4000	256
NER	CoNLL2003	F1	20	300	384
Generative	Math10K	-	3	-	256
	GSM8K	Accuracy	-	80	256
	SVAMP	Accuracy	-	80	256
	MultiArith	Accuracy	-	80	256
	AddSub	Accuracy	-	80	256
	AQuA	Accuracy	-	80	256
	SingleEq	Accuracy	-	80	256

(b) Task specific hyper-parameters

Table 7: All the hyper-parameters used in the paper.