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

Aradhye Agarwal[*] **Suhas Kamasetty Ramesh**[*]
Ayan Sengupta[*] **Tanmoy Chakraborty**
Indian Institute of Technology Delhi, India
aradhyeagarwal@gmail.com, 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. Selective-PEFT, a class of parameter-efficient fine-tuning (PEFT) methodologies, aims to mitigate these computational challenges by selectively fine-tuning only a small fraction of the model parameters. Although parameter-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 selected using different importance heuristics, failing to capture parameter importance dynamically and often leading to suboptimal performance. We introduce $\texttt{ID}^3$, a novel selective-PEFT method that calculates parameter importance continually, and dynamically unmasks parameters by balancing exploration and exploitation in parameter selection. Our empirical study on 16 tasks spanning natural language understanding, mathematical reasoning, and summarization demonstrates the effectiveness of our method compared to fixed-masking selective-PEFT techniques. We analytically show that $\texttt{ID}^3$ reduces the number of gradient updates by a factor of two, enhancing computational efficiency. Since $\texttt{ID}^3$ is robust to random initialization of neurons and operates directly on the optimization process, it is highly flexible and can be integrated with existing additive and reparameterization-based PEFT techniques such as Adapters and LoRA, respectively.[1]

1 Introduction

Pretrained large language models (LLMs) (Devlin et al., 2019; 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. In order to adapt these models to specific downstream tasks, fine-tuning on task-specific datasets is commonly employed to impart specialized domain knowledge. While larger models such as Qwen (Yang et al., 2024) and LLaMA (Touvron et al., 2023) enable promising alternatives like in-context learning (ICL), which allows quick task adaptation without gradient-based updates, recent research (Liu et al., 2022) suggests that ICL often underperforms compared to fine-tuning in terms of downstream performance and efficiency. As a result, parameter-efficient fine-tuning (PEFT) methods have gained prominence, offering a more practical balance between performance and computational cost when adapting large models to specific tasks.

Parameter-efficient fine-tuning (PEFT) aims to enhance the parameter, memory, and compute efficiency of model fine-tuning by performing low-rank or sparse updates instead of dense updates, as is typical in full fine-tuning (FFT). Additive PEFT methods (Houlsby et al., 2019; Pfeiffer et al., 2020; Chen et al., 2023; Lei et al., 2023) introduce additional trainable parameters to the frozen pretrained model. In contrast, reparameterization-based PEFT techniques (Hu et al., 2021; He et al., 2022; Yang et al., 2023; Liu et al., 2024) 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; Lawton et al., 2023), another class of PEFT techniques, use different heuristics to select a subset of parameters within the pretrained models for fine-tuning. The heuristic function assigns positive real-valued

[*]Equal contribution.

[1]Code is available at `https://github.com/Aradhye2002/selective-peft-toolkit`.

Transactions of the Association for Computational Linguistics, vol. 13, pp. 1767–1788, 2025. https://doi.org/10.1162/TACL.a.59
Action Editor: Kallirroi Georgila. Submission batch: 1/2025; Revision batch: 4/2025; Published 12/2025.

importance to each parameter in the model, while a suitable selection strategy determines which parameters to choose for fine-tuning based on the predicted importance. For instance, Diff Pruning (Guo et al., 2020) uses the change in parameter magnitude to assess the parameter importance, whereas Fish Mask (Sung et al., 2021) uses a gradient-based Fisher importance heuristic function. Most of these selective-PEFT techniques identify and fine-tune only a static set of top-B parameters from the entire parameter pool, where B is a fixed and predefined budget. Incorrect allocation of this budget can detrimentally impact the fine-tuned model's performance due to the suboptimal selection of parameters, either by including non-essential or excluding critical ones. The parameter selection strategies for these existing selective-PEFT techniques can be broadly classified into static (static-S) and repeated (repeat-S). These two strategies represent opposite extremes: Static-S is pure exploitation (reusing the same parameters throughout), whereas repeat-S is pure exploration (choosing a fresh set of parameters at each step). A majority of the existing selective-PEFT methods use static-S selection, and these exploitation-only methods often fail to select the optimal parameters for a given task. On the other hand, repeat-S-based PEFT methods often overshoot the target budget and perform well only for very small budgets.

To address these issues, we introduce a novel selection strategy called increment-S, which balances the exploration and exploitation strategies adopted in repeat-S and static-S, respectively. We analytically show that incremental parameter selection is computationally more efficient and 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, increment-S performance exceeds existing baselines. We also propose a new **D**ynamic magnitu**D**e and gra**D**ient-based heuristic (*a.k.a.* $\mathtt{D}^3$), which combines the benefits of magnitude and gradient-based parameter importance heuristics. Our proposed method, increment-$\mathtt{D}^3$ ($\mathtt{ID}^3$), can be easily integrated into any neural module and sparsify additive and reparameterization-based modules of pretrained models. Existing static-S PEFT techniques do not exhibit this property as they fail to assess parameter importance for randomly initialized untrained parameters.

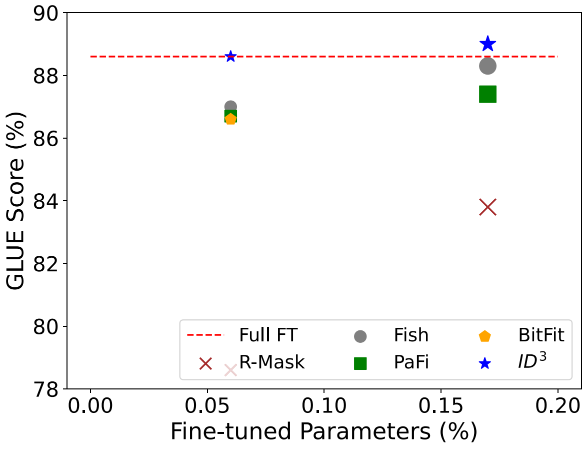


Figure 1: Comparison of different selective-PEFT methods–Full fine-tuning (Full FT), Random masking (R-Mask), Fish (Sung et al., 2021), BitFit (Zaken et al., 2021), PaFi (Liao et al., 2023), and $\mathtt{ID}^3$ on GLUE benchmark. Marker size denotes the number of trainable parameters. Detailed results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

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, $\mathtt{ID}^3$ outperforms other selective-PEFT baselines by a margin of 1.5% with the pretrained DeBERTa-v3 (He et al., 2021b) backbone. With only 0.17% of trainable parameters (320K), $\mathtt{ID}^3$ beats the fully fine-tuned DeBERTa-v3 model with a margin of 0.45% on average (cf. Figure 1). We explore $\mathtt{ID}^3$ with LoRA (Hu et al., 2021) fine-tuned LLaMA-7B (Touvron et al., 2023) and Qwen-2.5 (Yang et al., 2024) backbone models on six mathematical reasoning tasks, where $\mathtt{ID}^3$ achieves 0.6% better accuracy than the baselines on zero-shot classification.

Our major contributions are listed below:

(1) We introduce a novel selective strategy, increment-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, $\mathtt{D}^3$, that combines the benefits of gradient and magnitude-based parameter importance functions. Together with the increment-S strategy, our proposed selective-PEFT

method ID^3 demonstrates a strong performance on various natural language understanding and generation tasks, even with highly sparse parameter updates.

(3) Our method produces progressively improved models across increasing budget levels, allowing users to balance budget and performance effectively.

(4) We provide an open-source toolkit integrating three selective-PEFT techniques, offering comprehensive support for selective methods not available in existing toolkits.

2 Related Work

This section highlights the representative works in three broad categories of PEFT strategies: *additive*, *reparameterization-based*, and *selective*.

Additive PEFT methods, such as Adapters (Houlsby et al., 2019; Pfeiffer et al., 2020), add additional neural components to the pretrained models. Due to their additive nature, these methodologies usually offer flexibility in multi-task fine-tuning setups, where the same pretrained model is used with different task-specific adapters. The earliest adapter technique (Houlsby et al., 2019) utilized the additive component in feed-forward networks and attention layers 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. Lei et al. (2023) proposed Conditional-Adapter, which selectively activates different adapters for different input tokens. Chen et al. (2023) came up with a Hadamard Adapter that introduces additional weight and bias parameters and performs element-wise multiplication and addition to the self-attention outputs.

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. However, LoRA applies a uniform rank across all added parameters, thereby assuming that all parameter matrices are equally important. To address this limitation, AdaLoRA (Zhang et al., 2023c) dynamically allocates the parameter budget among the additional weight matrices with singular value decomposition of the ΔW 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. More recent methods like DyLoRA (Valipour et al., 2023), LoRA+ (Hayou et al., 2024), and DoRA (Liu et al., 2024) aim at improving the training efficiency and adaptability of low-rank adaptation on downstream tasks.

Selective-PEFT strategies generate a sparse mask $M \in \{0, 1\}^{|W|}$ corresponding to each weight matrix W in the pretrained 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 parameter. In contrast, Diff Pruning (Guo et al., 2020) evaluates the absolute parameter changes across successive training phases, pruning those with the smallest magnitude. Determining the magnitude of parameter change requires significant computational and storage costs, equivalent to full fine-tuning of the model. To alleviate these computational burdens, Sung et al. (2021) and Das et al. (2023) utilized the empirical Fisher importance matrix for selective fine-tuning. To avoid the cost of measuring parameter importance, Liao et al. (2023) proposed PaFi, which assesses the significance based on the absolute magnitude of the parameters and retains only ones with least magnitude. Unlike earlier methods that modify the pretrained model directly, He et al. (2022) proposed SparseAdapter, a novel approach that merges with existing adapter-based techniques to sparsify an adapter fine-tuned model, enhancing the efficiency of PEFT. On a similar attempt, Zhang et al. (2023b) proposed LoRAPrune to combine LoRA with structured pruning to iteratively and progressively reduce model size while maintaining performance.

Our proposed ID^3 method distinguishes itself from current selective-PEFT methods by progressively selecting the parameters throughout fine-tuning, thereby capturing the change in parameter importance during the training process. Additionally, ID^3 can choose model checkpoints with incremental budgets, which is not possible with existing selective-PEFT methods. ID^3 also leverages both the magnitude and gradient of parameters, which can be efficiently computed using any automatic differentiation tool (Baydin et al., 2018), thereby avoiding extra computational delays.

3 Methodology

Motivated by the key challenges of the existing selective-PEFT methodologies highlighted in Sections 1 and 2, we propose ID^3, an iterative approach for calculating the parameter importance and incrementally selecting the top parameters for each training iteration. 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 itself as the tensor parameter. For instance, a tensor parameter in a BERT (Devlin et al., 2019) 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 selective-PEFT method as a heuristic function combined with a selection strategy. We identify three common selection strategies: (1) static-S, where the initial set of parameters, selected according to the heuristic, is reused throughout training; (2) repeat-S, where we use the heuristic repeatedly at each training step to find a (potentially) new selected set, and (3) increment-S, where we accumulate the selected set over the training iterations, guided by the heuristic. These selection strategies are illustrated in Figure 2. Existing selective-PEFT methods use static-S, ID^3 uses increment-S, while repeat-S is treated as a baseline for comparison.

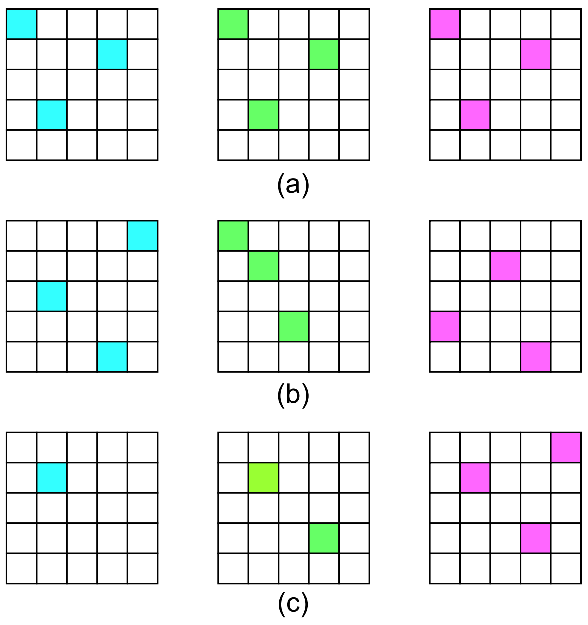


Figure 2: Different parameter selection strategies. Here $B = 3$ represents the budget while $T = 3$ represents the training steps. **(a) Static-S** strategy, where B number of parameters are chosen initially and used in all future training steps. **(b) Repeat-S**, where B number of fresh parameters are chosen according to the heuristic at each training step. **(c) Increment-S**, where $k = \frac{B}{T}$ parameters are chosen at each training step as per the heuristic.

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 θ, we calculate an importance function $f : \mathbb{R}^2 \to [0, \infty]$ that measures a real-valued importance for each parameter given its value θ^i and the gradient, ∇_{θ^i}. Formally, we define the parameter importance function (also referred to as the heuristic function):

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

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. We also note that such a functional form is general enough to represent both the PaFi and Fish metrics by varying the value of exp ($exp = 0$ reduces D^3 to Fish, while $exp = \infty$ converts it to PaFi). The following theorem also 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 θ, the Fisher information matrix $\mathcal{I}(\theta)$ is the variance

$$\mathbb{E}_{x,y}\left[\left(\frac{\partial}{\partial\theta}\log f(x,y;\theta)\right)^2\right] - \mathbb{E}_{x,y}\left[\left(\frac{\partial}{\partial\theta}\log f(x,y;\theta)\right)\right]^2 \tag{2}$$

Fisher information measures the amount of information the random variable x carries about the unknown model parameter θ 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 $\mathbb{E}_{x,y}\Big[\mathcal{H}(\theta)\Big]$.

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

$$\mathbb{E}_{x,y}\left[\left(\frac{\partial}{\partial\theta}\log f(x,y;\theta)\right)\right]$$
$$= \int_x\int_y \frac{\frac{\partial}{\partial\theta}f(x,y;\theta)}{f(x,y;\theta)} f(x,y;\theta)p(x)\cdot dx\cdot dy$$
$$= \frac{\partial}{\partial\theta}\int_x\int_y f(x,y;\theta)p(x)dx\cdot dy = \frac{\partial}{\partial\theta}\cdot 1 = 0$$

Therefore,

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

Using Jensen's inequality, we get,

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

Hence, for $\epsilon \geq 1$ and $exp \geq 0$, $\mathcal{I}(\theta) \geq \left(\mathbb{E}_{x,y}\Big[\mathcal{H}(\theta)\Big]\right)^2$. Therefore, Theorem 1 justifies that maximizing $\mathcal{H}(\theta^i)$ indirectly maximizes Fisher importance.

3.2 Incremental Parameter Updates

Suppose we want to fine-tune a pretrained 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 $\frac{N-B}{N}$ is called *sparsity* 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 $\Lambda_t = \emptyset$. 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 Λ_t such that

$$\max_{\lambda_t}\min_{\theta^i\in\lambda_t}\{\mathcal{H}(\theta^i)\} \text{ s.t. } |\lambda_t| = u_t \tag{3}$$

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 $\mathcal{L}\Big(y, p_{\theta_{(t)}}(\cdot|x)\Big)$, while during the backward pass, the gradients $\nabla_{\theta_{(t)}}$ are set to zeros for parameters not in the unmask set Λ_t, obtaining $\tilde{\nabla}_{\theta_t}$. Formally,

$$\tilde{\nabla}_{\theta_t^i} = \begin{cases}\nabla_{\theta_t^i}, & \text{if } \theta_t^i \in \Lambda_t\\ 0 & \text{otherwise}\end{cases} \tag{4}$$

Finally, the parameters are updated using the filtered gradients $\tilde{\nabla}_{\theta_{(t)}}$. Algorithm 1 formalizes the ID^3 incremental parameter update procedure. With the incremental parameter selection and updates, the total number of parameter updates can be calculated as

$$U_{dynamic} = \sum_{t=0}^{T-1}\sum_{i=0}^{t} u_i$$

For the uniform unmasking scheduler,

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

Algorithm 1 Incremental parameter updates

```
Require: Unmasking scheduler $\{u_t\}_{t=1}^{T}$, number of training steps $T$, trainable model $\theta_{(0)}$, training dataset $(X, Y)$, learning rate $\eta$
  $t \leftarrow 0$
  $\Lambda_0 \leftarrow \emptyset$
  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 3
      $\Lambda_{t+1} \leftarrow \Lambda_t \cup \lambda_t$
      Update parameter gradients $\tilde{\nabla}_{\theta_{(t)}}$ using Equation 4
      Perform parameter update $\theta_{(t+1)} \leftarrow \theta_{(t)} + \eta\tilde{\nabla}_{\theta_{(t)}}$
      $t \leftarrow t + 1$
  end while
```

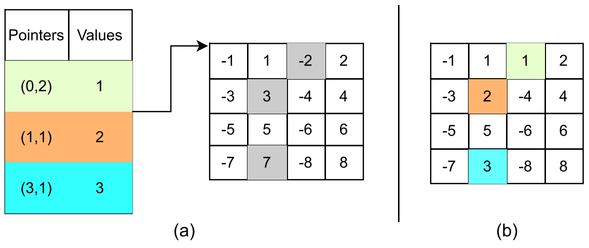


Figure 3: **(a, right):** Tensor parameter in the pre-trained model. **(a, left):** Table storing pointers and corresponding values of the scalar parameters updated during fine-tuning. **(b):** Final tensor parameter in the fine-tuned model, where old scalar values are replaced with updated ones.

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

$$U_{static} = \sum_{t=0}^{T} B = T \cdot B$$

Hence,

$$U_{dynamic} = \frac{U_{static}}{2} (\text{when } T \gg 1)$$

Therefore, incremental selection with a uniform schedule 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, which can be stored using a 32-bit unsigned integer. Each updated model parameter, on the other hand, can be stored using 64-bit double-precision floating-point numbers. Therefore, we can reduce the space complexity required to just $\mathcal{O}(2 \times 32 \times B + 64 \times B) = \mathcal{O}(B)$. While loading, we can use these pointers (stored in the form of tensors) to index into the tensor parameters and replace the pretrained parameters with the stored ones that 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 pretrained language models. On token classification, we experiment with the named entity recognition (NER) task using the CoNLL-2003 dataset (Tjong Kim Sang and DeMeulder, 2003). For these nine tasks, we fine-tune the model using the training splits and evaluate its performance on the validation splits.

For text generation we consider the CNN/Daily Mail summarization (Hermann et al., 2015; Nallapati et al., 2016) task and six mathematical reasoning tasks: GSM8K, SVAMP, MultiArith, AddSub, AQuA, and SingleEq. For the summarization task we train and evaluate on the training and dev splits of the original dataset. For mathematical reasoning tasks, we fine-tune the models on the Math10K dataset, as curated by Hu et al. (2023), and evaluate them on the test splits of the datasets above. We use 10% of the training data for validation. Detailed descriptions of these datasets and tasks are provided in Section 8.1 and Table 10 of Appendix.

4.2 Models

For NLU and NER tasks, we use the pretrained encoder-only DeBERTa-v3-base (He et al., 2021b) and RoBERTa-base (Liu et al., 2019) models as the backbone, while for summarization, we use the pretrained T5-small (Raffel et al., 2020) model. For math reasoning tasks, we use pretrained LLaMA-7B (Touvron et al., 2023), Qwen-2.5 (Yang et al., 2024), and MobileLLaMA-2.7B (Chu et al., 2023) models. All the pretrained 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*. We use

Budget	Method	MNLI-m	MNLI-mm	QQP	QNLI	SST-2	STS-B	CoLA	MRPC	RTE	Avg
184M	Full-FT	90.21	90.32	91.98	94.1	96.16	90.89	70.65	89.71	83.21	88.58
103K	R-Mask	82.74	83.63	86.11	88.04	92.77	80.66	60.24	76.04	57.22	78.61
	Fish	87.91	88.11	87.35	92.09	95.10	91.30	68.12	90.01	83.31	87.03
	PaFi	87.80	87.99	88.97	93.20	95.53	89.12	67.67	89.34	80.69	86.70
	BitFit	88.10	88.03	88.61	92.83	95.13	89.11	68.75	89.10	79.88	86.61
	ID^3	**89.33**	**89.59**	**89.84**	**93.62**	**95.56**	**91.97**	**70.49**	**90.81**	**85.83**	**88.56**
320K	R-Mask	87.32	87.54	88.47	91.35	94.67	85.61	64.84	81.68	72.38	83.76
	Fish	88.94	89.66	88.73	93.93	95.53	91.92	69.25	90.57	86.64	88.35
	PaFi	89.15	89.27	89.97	93.71	95.84	89.84	68.39	90.20	80.60	87.44
	ID^3	**89.58**	**89.73**	**90.31**	**94.03**	**95.90**	**91.97**	**71.46**	**91.12**	**87.19**	**89.03**
Wilcoxon statistic (p-value)		**465.0 (1e-5)**	**474.0 (1e-5)**	**459.5 (5e-5)**	**400.5 (1e-3)**	234.5 (0.71)	**402.0 (4e-3)**	**525.5 (1e-9)**	**389.5 (9e-5)**	**359.0 (1e-2)**	**493.0 (8e-7)**

Table 1: Mean performance of selective-PEFT methods on GLUE tasks with DeBERTa-v3. BitFit is evaluated only at the 103K budget, corresponding to DeBERTa-v3's 103K bias parameters. R-mask denotes a random static mask baseline. The best-performing method within each budget group is shown in **bold**. Standard deviations are provided in Table 12a of Appendix 8.3. We calculate the Wilcoxon statistic (and the associated p-value) for each GLUE task to assess the statistical significance of the improvement shown by ID^3 over the baselines. **Bold** indicates the tasks where p-value < 0.05.

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) which involves fine-tuning only the bias terms in a pretrained model; (2) PaFi (Liao et al., 2023) which selects the pretrained parameters with the smallest magnitude and trains only these parameters during fine-tuning; (3) ID^3.

The toolkit allows integration of these selective-PEFT methods into the original pretrained 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 in a memory efficient manner, enabling end-to-end training and evaluation workflows.

Additional details and hyperparameters for reproducing the results are provided in Section 8.2 and Table 11 of the Appendix. All experiments are conducted on Nvidia A100 and A6000 GPUs.

5 Results

This section presents the results of our exhaustive experiments on text classification, token classification, and text generation.

5.1 Text Classification

We report the results on GLUE tasks in Table 1. ID^3 achieves an average score of 89.03% with a budget of 320K, surpassing the best-performing baseline (Fish) by over 0.6%. Interestingly we observe that ID^3 outperforms even the FFT baseline (88.58%). A similar comparison holds at the smaller budget level of 103K, with ID^3 outperforming other selective baselines by more than 1%. We perform paired Wilcoxon tests[2] between the results obtained by ID^3 and the best baselines (for each task) across all the budgets to compute the Wilcoxon statistic. At an overall level, we obtain a Wilcoxon statistic of 103.0 with a p-value of 0.04, indicating the statistical significance of the competitive performance of ID^3. ID^3 outperforms existing baselines with statistical significance on 8 of 9 GLUE tasks.

We further evaluate the effectiveness of ID^3 with other adapters integrated with pretrained 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^3. With a budget of 320K (sparsity 76%), ID^3 matches full LoRA fine-tuning with an average of 88.76%. Interestingly, LoRA sparsified with both ID^3 and PaFi beats the dense LoRA model on 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 phenomenon as shown in Table 3. With a budget of only 320K (sparsity 96%), ID^3 can improve the performance of an adapter-integrated RoBERTa-base by a margin of

[2]Additional details regarding the significance testing methodologies are presented in Appendix 8.5.

Budget	Method	MNLI-m	MNLI-mm	QQP	QNLI	SST-2	STS-B	CoLA	MRPC	RTE	Avg
1.33M	LoRA (r = 8)	90.47	90.46	91.95	93.76	95.57	91.86	69.73	89.71	85.32	88.76
320K	PaFi + LoRA (r = 8)	89.95	89.89	91.20	94.09	**95.99**	90.90	**70.22**	90.01	83.87	88.46
320K	ID^3 + LoRA (r = 8)	**90.11**	**90.06**	**91.48**	**94.15**	95.70	**91.58**	68.83	**90.50**	**86.46**	**88.76**
	Wilcoxon statistic	83.0	95.0	82.0	61.0	0.0	**134.0**	1.0	**77.0**	**131.5**	**103.0**
	(p-value)	(0.09)	(0.09)	(0.25)	(0.65)	(0.99)	**(5e-5)**	(0.99)	**(0.01)**	**(1e-4)**	**(0.04)**

Table 2: Performance of PaFi and ID^3 with LoRA+pretrained DeBERTa-v3 on GLUE tasks. Standard deviations are reported in Table 12b (Appendix 8.3). The average improvement of ID^3 over PaFi is statistically significant, but results remain inconclusive for six of nine tasks (p-value ≥ 0.05).

Budget	Method	STS-B	CoLA	MRPC	RTE	Average
8M	Pfeiffer	90.78	59.05	89.21	76.53	78.89
320K	SparseAdapter + Pfeiffer	**90.88**	58.95	89.41	77.03	79.07
320K	ID^3 + Pfeiffer	90.71	**59.84**	**89.95**	**79.42**	**79.98**

Table 3: Performance of ID^3 compared with SparseAdapter (He et al., 2022) on Pfeiffer adapter (Pfeiffer et al., 2020) applied to pretrained RoBERTa (Liu et al., 2019). A Wilcoxon statistic of 9.0 highlights that ID^3 outperforms SparseAdapter, however, a p-value of 0.12 indicates that the results cannot be concluded statistically significant under a significance level of 0.05.

Budget	Full-FT	Fish	PaFi	BitFit	ID^3	R-Mask
103K	–	95.26	94.40	93.85	**95.55**	70.15
320K	–	95.93	95.42	–	**96.04**	89.93
184M	96.62			–		

Table 4: Mean performance of selective methods with DeBERTa-v3 on NER at different budgets. As the DeBERTa model has 103K bias terms, BitFit is only run with the 103K budget. Corresponding standard deviations are reported in Table 12c of Appendix 8.3. A Wilcoxon statistic of 385.0 with p-value 0.01 indicates the statistical significance of improvement shown by ID^3 over the baselines.

1.09%. SparseAdapter, another popular sparsification technique for adapters, falls short by 0.91% compared to ID^3.

5.2 Token Classification

The CoNLL benchmark results in Table 4 highlight ID^3 as a top-performing PEFT method, achieving an F1 score of 95.55% with only 103K parameters surpassing Fish (95.26%) and PaFi (94.40%). With a larger 320K parameter budget, ID^3 improves to 96.04%, approaching FFT's baseline of 96.62% (184M parameters).

Budget	Method	Rouge-1	Rouge-2	Rouge-L
60M	FFT	41.29	18.90	29.19
100K	PaFi	40.15	18.03	28.49
	ID^3	**40.43**	**18.44**	**28.76**
320K	PaFi	40.75	18.57	28.83
	ID^3	**40.91**	**18.73**	**28.98**
1M	PaFi	41.16	18.79	29.09
	ID^3	**41.17**	**18.85**	**29.17**
	Wilcoxon statistic	6.0	6.0	6.0
	(p-value)	(0.13)	(0.13)	(0.13)

Table 5: Performance of ID^3 and PaFi with T5-small on summarization.

This demonstrates ID^3's efficiency and robustness as a highly effective alternative to full fine-tuning.

5.3 Text Generation

We evaluate ID^3 along with the other selective baselines on two text generation tasks which include abstractive summarization and mathematical reasoning.

5.3.1 Summarization

The results of T5-small on the CNN/Daily Mail summarization task in Table 5 show that fine-tuning all 60M parameters (FFT) achieves the highest performance with Rouge-1 of 41.29, Rouge-2 of 18.90, and Rouge-L of 29.19. Among the selective methods, ID^3 consistently outperforms PaFi across all parameter budgets. At 100K parameters, ID^3 achieves Rouge scores of 40.43/18.44/28.76, improving over PaFi by 0.28/0.41/0.27 points. At 320K, ID^3 improves to 40.91/18.73/28.98, surpassing PaFi by 0.16/0.16/0.15. At 1M, ID^3 scores 41.17/18.85/29.17, slightly outperforming PaFi. While FFT remains superior, ID^3 demonstrates its efficiency and robustness as an effective alternative under constrained parameter budgets.

5.3.2 Mathematical Reasoning

Table 6 presents the results of various mathematical reasoning tasks. LLaMA-7B fine-tuned with

Model	Budget	Method	AddSub	MultiArith	SingleEq	GSM8K	AQuA	SVAMP	Avg.
LLaMA-7B	56M	LoRA (r = 32)	81.3	95.5	81.7	34.1	17.7	46.7	59.5
	3.5M	LoRA (r = 2)	78.2	**96.7**	76.6	**35.3**	**16.9**	44.9	58.1
	3.5M	PaFi + LoRA (r = 32)	78.7	92.3	76.8	33.9	**16.9**	43.2	57.0
	3.5M	$\texttt{ID}^3$ + LoRA (r = 32)	**80.7**	95.8	**79.3**	34.3	15.7	**45.7**	**58.6**
Qwen-7B	54M	LoRA (r = 32)	94.4	98.2	97.6	76.9	34.6	85.8	81.3
	3.4M	LoRA (r = 2)	**93.9**	98.3	96.4	76.4	31.9	86.8	80.6
	3.4M	PaFi + LoRA (r = 32)	91.1	**99.0**	**97.0**	**78.5**	**37.8**	85.8	**81.5**
	3.4M	$\texttt{ID}^3$ + LoRA (r = 32)	93.6	98.5	95.1	77.9	37.0	**87.1**	**81.5**
Qwen-3B	40M	LoRA (r = 32)	92.1	98.5	95.9	71.9	34.2	81.5	79.0
	2.5M	LoRA (r = 2)	**92**.9	97.5	94.9	70.8	34.6	**85.1**	79.3
	2.5M	PaFi + LoRA (r = 32)	90.9	97.8	**96.2**	70.6	36.2	83.9	79.3
	2.5M	$\texttt{ID}^3$ + LoRA (r = 32)	92.6	**98.2**	95.9	**71.5**	**37.4**	83.9	**79.9**
Qwen-1.5B	25M	LoRA (r = 32)	90.4	98.2	96.6	65.8	36.6	75.3	77.2
	1.5M	LoRA (r = 2)	**91.9**	**98.2**	95.5	62.8	31.1	80.9	76.7
	1.5M	PaFi + LoRA (r = 32)	89.4	96.7	**95.9**	**64.5**	32.3	78.4	76.2
	1.5M	$\texttt{ID}^3$ + LoRA (r = 32)	91.6	97.8	93.7	62.6	**34.6**	**81.0**	**76.9**
	Wilcoxon statistic (p-value)		4.0 (0.69)	4.0 (0.69)	4.0 (0.69)	4.0 (0.69)	6.0 (0.44)	6.0 (0.44)	10.0 (0.06)

Table 6: Results on mathematical reasoning obtained from LLaMA and Qwen with LoRA fine-tuning. We report the Wilcoxon statistic alongside the associated p-value for highlighting the statistical significance of the results.

Model	Budget	Method	AddSub	MultiArith	SingleEq	GSM8K	AQuA	SVAMP	Avg.
MobileLLaMA-2.7B	2.7B	FFT	79.7	95.8	82.2	33.3	18.1	31.8	56.8
	2.7M	PaFi	46.1	66.5	46.6	11.1	**18.7**	23.1	35.4
	2.7M	$\texttt{ID}^3$	**47.1**	**67.8**	**48.4**	**11.9**	17.3	**25.0**	**36.3**
	1.3M	PaFi	30.1	36.2	30.7	8.1	**21.2**	17.5	24.0
	1.3M	$\texttt{ID}^3$	**35.2**	**57.8**	**41.1**	**8.6**	15.7	**22.0**	**30.1**

Table 7: Results of MobileLLaMA-2.7B with full fine-tuning on mathematical reasoning tasks. A Wilcoxon statistic of 88.0 with a p-value of 0.01 indicates the statistical significance of $\texttt{ID}^3$'s improvement over PaFi.

LoRA (r = 32) achieves a strong baseline average score of 59.5%. Notably, even when the parameter budget is reduced to 3.5M, LoRA (r = 2) maintains robust performance with an average of 58.1%, excelling in MultiArith (96.7%) but showing minor drops on other tasks compared to the 56M setting. Applying $\texttt{ID}^3$ to LoRA (r = 32) yields a slightly higher average score of 58.6%, outperforming LoRA (r = 2) with the same parameter budget. This setup delivers strong results on AddSub (80.7%) and SingleEq (79.3%), suggesting that sparsifying higher-rank LoRA modules enhances performance. PaFi combined with LoRA achieves an average score of 57.0%, with its best result in MultiArith (92.3%), though it generally trails behind both full-rank LoRA and $\texttt{ID}^3$ in other tasks. On Qwen-7B, both PaFi + LoRA and $\texttt{ID}^3$ + LoRA reach an average score of 81.5, marginally surpassing LoRA (r = 32) at 81.3. Similar trends hold for Qwen-3B and Qwen-1.5B, where $\texttt{ID}^3$ + LoRA consistently matches or exceeds the performance of PaFi + LoRA while maintaining parameter efficiency. Specifically, $\texttt{ID}^3$ leads in reasoning-heavy tasks like GSM8K (71.5 vs. 70.6 on Qwen-3B) and AQuA (34.6 vs. 32.3 on Qwen-1.5B), while PaFi performs slightly better on MultiArith (99.0 vs. 98.5 on Qwen-7B) and SingleEq (96.2 vs. 95.9 on Qwen-3B). Overall, $\texttt{ID}^3$ demonstrates greater robustness and generalization across tasks, particularly under constrained parameter budgets. Combining $\texttt{ID}^3$ or PaFi with LoRA enhances task performance by balancing efficiency with accuracy.

Table 7 highlights the performance of $\texttt{ID}^3$ and PaFi when used directly on the pretrained MobileLLaMA-2.7B model. The fully fine-tuned MobileLLaMA model achieves 56.8% accuracy on average. With a 2.7M budget (0.1% of the entire model), $\texttt{ID}^3$ recovers 64% of the performance (achieving 36.3% accuracy), whereas PaFi recovers 62% of the average performance. Surprisingly, on more challenging tasks like AQuA and SVAMP, the recovery is higher with both the methods, 87% with PaFi and 83% with $\texttt{ID}^3$.

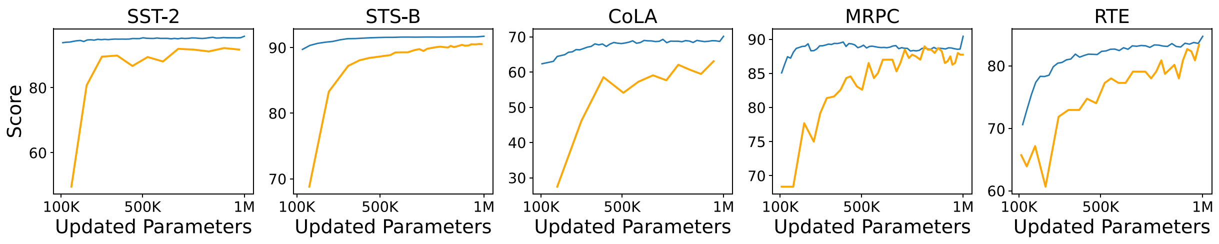


Figure 4: Performance of D^3 with increment-S (blue line) and repeat-S (orange line) parameter selection.

At a lower budget, the recovery drops for both methods, with ID^3 remaining more robust (recovery 53%) than PaFi (recovery 42%). These results indicate that even for larger models (over 1B parameters), full fine-tuning can be avoided with selective alternatives, incurring only slight drops in performance.

6 Analysis

Here, we study different aspects of ID^3 and their importance in the efficient fine-tuning of LLMs.

6.1 Importance of Incremental Selection

We explore a variant of ID^3 that uses the repeat-S strategy instead of increment-S. As shown in Figure 4, the increment-S strategy works better for almost all budgets between 100K (sparsity 99.9%) and 1M (sparsity 98.8%). Although the performance gap between increment-S and repeat-S reduces for higher budgets, the practical application of the repeat-S strategy remains restricted due to its inferior performance at lower budgets. For a fixed budget, repeat-S typically updates more unique parameters in the model (due to the aggressive exploration strategy at each step) than increment-S. Therefore, it is prone to updating unimportant parameters, leading to lesser performance. Further, for tasks like MRPC and RTE, with limited training samples, repeat-S performance fluctuates across consecutive steps. ID^3, on the other hand, minimizes unnecessary parameter updates, achieving a better overall performance.

6.2 Importance of the D^3 Metric

Figure 5 illustrates the performance of the PaFi heuristic with increment-S selection. On average, using the PaFi heuristic results in a 5% drop in performance compared to the D^3 metric, with the largest drop being 12% on the RTE task. This underwhelming performance highlights the critical role of the D^3 metric in determining pa-

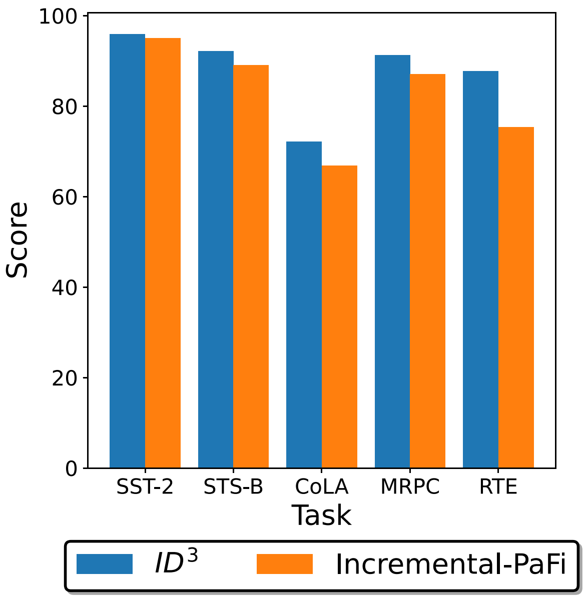
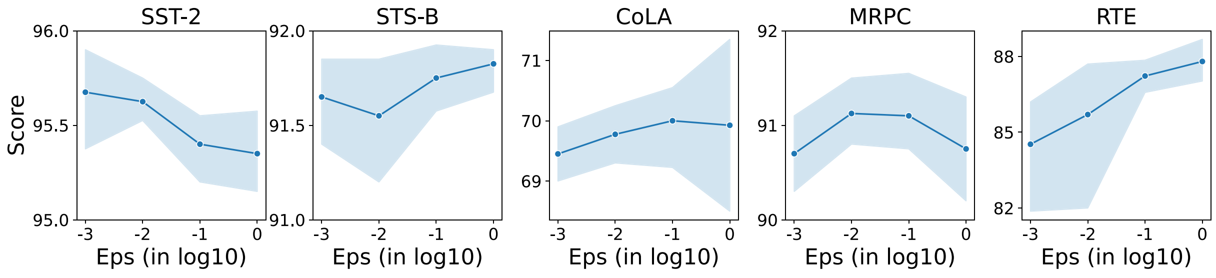


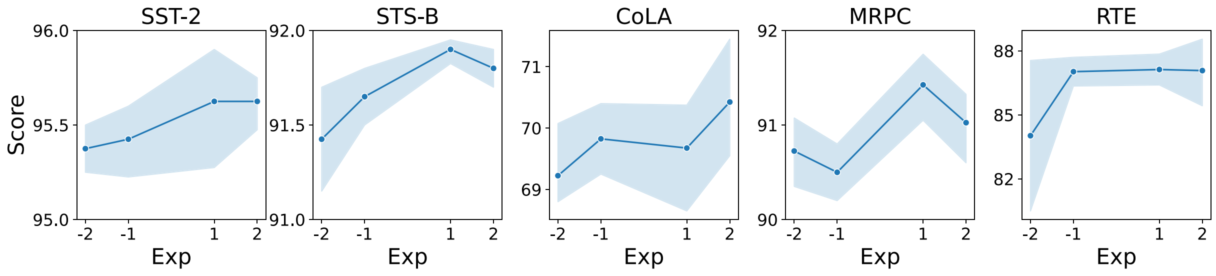
Figure 5: Performance of ID^3 and PaFi with increment-S strategies with DeBERTa-v3.

rameter importance during fine-tuning. Unlike D^3, the PaFi metric relies solely on the magnitude of parameters to assess importance, potentially overlooking their relative significance towards the task-specific learning objective. This limitation becomes more pronounced when paired with an incremental scheduling strategy. In contrast, D^3 incorporates both magnitude and gradient information, capturing both the absolute and relative importance of parameters, thereby leading to superior performance.

To further understand how different components of D^3 work, we perform an ablation study on ϵ and exp. Figure 6a highlights that the best performance is achieved typically with $\epsilon \in \{0.1, 1\}$. Lower values of ϵ have a less smoothing effect, preventing parameters with low gradients from being unmasked unfairly. An interesting trend is also observed with exp (cf. Figure 6b), where $exp \in \{1, 2\}$ consistently performs better than $\{-2, -1\}$. It is, however, worth noting that these



(a) Performance of $\texttt{ID}^3$ with different ϵ



(b) Performance of $\texttt{ID}^3$ with different exp

Figure 6: Performance of $\texttt{ID}^3$ under different ϵ and exp values with DeBERTa-v3 backbone model.

performance improvements are statistically insignificant (cf. Table 8). Our one-way ANOVA test highlights that the exact values of ϵ and exp do not change the overall performance of $\texttt{ID}^3$. These results emphasize the robustness of $\texttt{ID}^3$ under different choices of ϵ and exp values, demonstrating that $\texttt{ID}^3$ does not require extensive tuning.

	SST-2	STS-B	CoLA	MRPC	RTE
ϵ	1.07 (0.40)	0.96 (0.46)	0.36 (0.83)	1.38 (0.28)	1.26 (0.33)
exp	1.15 (0.37)	**3.02 (0.05)**	0.51 (0.73)	0.82 (0.53)	0.98 (0.44)

Table 8: One-way ANOVA test results for assessing the importance of ϵ and exp values. We report the F-statistics and the p-values. Statistically significant results (p-value $\leq$ 0.05) are shown in **bold**.

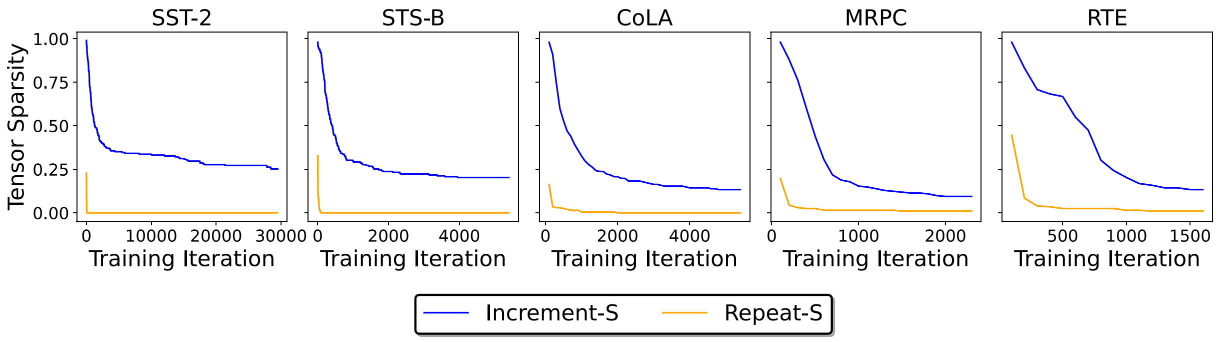
6.3 Sparsity and Importance with $\texttt{ID}^3$

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^i such that $P^i \cap \Lambda_t = \emptyset$. Figure 7a highlights the tensor sparsity for $\texttt{ID}^3$ with increment-S and repeat-S selection at different training iterations. For all the tasks, tensor sparsity remains close to one for $\texttt{ID}^3$ 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 exploitation from the same tensor parameters. A similar behavior is also observed with repeat-S parameter selection. However, with this approach, the tensor sparsity remains much lower, as this selection method exceeds the budget and can potentially fine-tune the entire model.

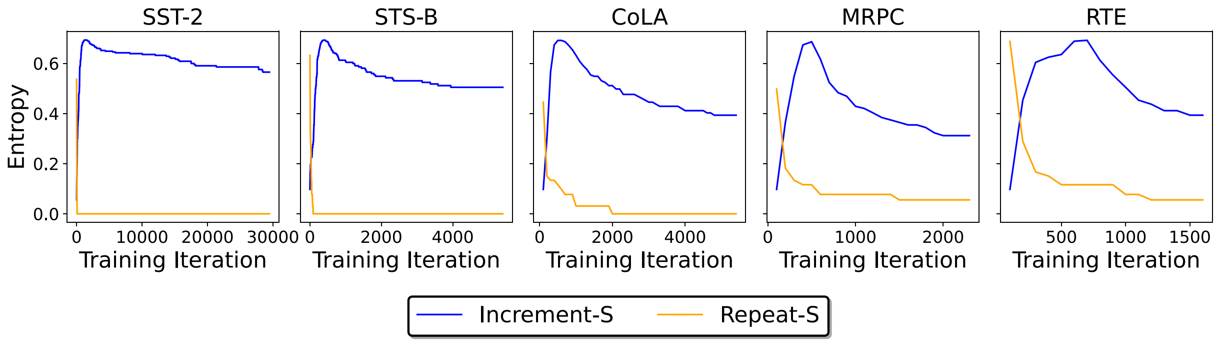
To understand how $\texttt{ID}^3$ impacts different tensor parameters in a model, we compute the selection probability for each tensor parameter P_j as $|P_j \cap \Lambda_T|/|P_j|$. Using this probability distribution over all the tensor parameters, we calculate the selection entropy of the fine-tuned model. A high entropy indicates uniform selection probability across different parameters, indicating uniform parameter importance. Figure 7b suggests that for increment-S, 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 strategy performs drastic exploration, unmasking most of the tensor parameters quickly and thereby reducing entropy rapidly.

6.4 Difference Between FFT and $\texttt{ID}^3$

We perform detailed analysis on the DeBERTa-v3 model on the STS-B task fine-tuned with FFT and $\texttt{ID}^3$. The primary objective of this analysis is to

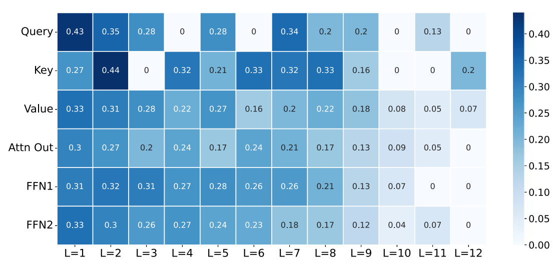


(a) Tensor Sparsity

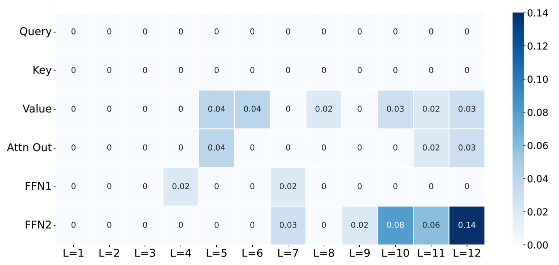


(b) Tensor Entropy

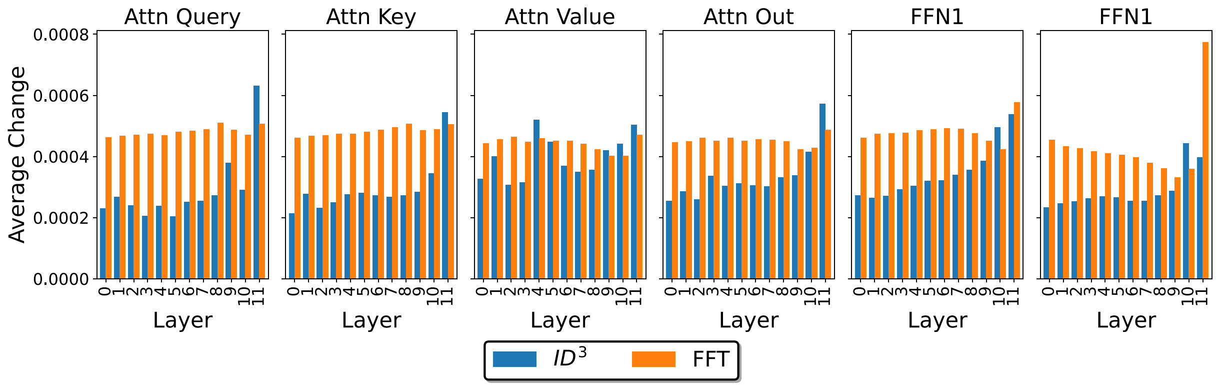
Figure 7: Tensor sparsity and entropy with increment-S and repeat-S selection strategies.



(a) Correlation between FFT and ID^3 parameters



(b) Importance of parameters unmasked in ID^3



(c) Parameter update magnitude in FFT and ID^3

Figure 8: FFT and ID^3 analysis on STS-B. (a) Delta change in parameter weight remains highly correlated between FFT and ID^3. (b) However, the parameters not selected in ID^3 (potentially unimportant) are also significantly updated with the FFT strategy. (c) At the tensor level, FFT updates the parameters with a higher magnitude than ID^3. However, unlike FFT, ID^3 incorporates tensor importance and updates the tensors parameter accordingly.

Method	Peak memory (GB)	Initialization (s)	Update (s)	Overall (s)
FFT	10.10	0.0	0.00	0.23
BitFit	10.29	0.05	0.00	0.24
PaFi	10.29	4.58	0.00	0.24
$\texttt{ID}^3$	12.92	2.30	0.10	0.33

Table 9: Computational complexity of selective methods with DeBERTa-v3 model. We report the peak GPU memory consumed (in GB), along with the time taken in seconds for mask initialization, mask update, and overall time during one optimization step.

gather more insight into the workings of $\texttt{ID}^3$ and how it behaves compared to full fine-tuning.

Figure 8a shows the Spearman correlation coefficient between the magnitude of parameter change with FFT and $\texttt{ID}^3$ on the intersecting parameters (*i.e.,* parameters updated by both $\texttt{ID}^3$ and FFT). A high correlation indicates that the parameters common to FFT and $\texttt{ID}^3$ have similar ordering in terms of importance. On the other hand, Figure 8b suggests that under FFT, even the non-overlapping parameters are also subjected to significant gradient updates. This trend highlights the inability of the FFT strategy to determine parameter importance during fine-tuning. Figure 8c shows the average change in parameter values after fine-tuning. On average, FFT makes more changes to the parameter values than $\texttt{ID}^3$, potentially also updating the unimportant parameters. However, it is worth noting that the magnitude of parameter updates under $\texttt{ID}^3$ varies between different modules and layers. Self-attention query and key matrices, often considered important for syntactic language understanding, are updated moderately with $\texttt{ID}^3$ compared to FFT. On the other hand, self-attention value and feed-forward modules that are responsible for capturing semantics and task-specific knowledge are subjected to higher updates with $\texttt{ID}^3$. Another interesting observation is that $\texttt{ID}^3$ makes more change to the later layers of the encoder backbone model, indicating the assignment of greater weight (and hence importance) toward these layers. These demonstrations support the literature (Jawahar et al., 2019; Clark et al., 2019) that has previously shown that the semantic understanding of language models tends to benefit most from the middle layers, while the upper layers contribute more to task-specific feature learning.

6.5 Efficiency Comparison

Table 9 compares GPU memory usage and execution time per step for FFT and selective methods. Selective methods like BitFit and PaFi have minimal overhead due to static-S strategies, with memory usage only slightly higher than FFT (10.29 vs. 10.10 GB). In contrast, $\texttt{ID}^3$'s incremental strategy, requiring additional tensor operations during mask updates, increases memory usage to 12.92 GB. BitFit's simple parameter selection is the fastest (0.05s), while PaFi's more complex logic takes twice the time, and $\texttt{ID}^3$, with continuous updates, has the slowest per-step execution. However, $\texttt{ID}^3$ has fixed time and memory costs per step, allowing these overheads to be amortized with larger batch sizes, thereby minimizing their relative impact on the computational cost of fine-tuning.

7 Conclusion

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

Limitations and Future Work While selective-PEFT methods do reduce the number of gradient updates (with $\texttt{ID}^3$ 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. In order 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. We also hope our work inspires further research into the mechanistic activation of selectively updated parameters to deepen the understanding of selective fine-tuning and improve explainability in LLMs.

Acknowledgments

We acknowledge the support of the IBM-IITD AI Horizons network. T. Chakraborty acknowledges the support of the Rajiv Khemani Young Faculty Chair Professorship in Artificial Intelligence. We also thank the reviewers and the TACL action editor for their thoughtful comments and suggestions. Their feedback greatly helped improve the clarity, rigor, and overall quality of the paper.

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, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 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. `https://doi.org/10.18653/v1/S17-2001`

Yuyan Chen, Qiang Fu, Ge Fan, Lun Du, Jian-Guang Lou, Shi Han, Dongmei Zhang, Zhixu Li, and Yanghua Xiao. 2023. Hadamard Adapter: An extreme parameter-efficient adapter tuning method for pre-trained language models. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 276–285. `https://doi.org/10.1145/3583780.3614904`

Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. 2023. A survey on deep neural network pruning-taxonomy, comparison, analysis, and recommendations. *arXiv preprint arXiv:2308.06767*.

Xiangxiang Chu, Limeng Qiao, Xinyang Lin, Shuang Xu, Yang Yang, Yiming Hu, Fei Wei, Xinyu Zhang, Bo Zhang, Xiaolin Wei, and Chunhua Shen. 2023. MobileVLM: A fast, strong and open vision language assistant for mobile devices. *arXiv preprint arXiv:2312.16886*.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does BERT look at? An analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286, Florence, Italy. Association for Computational Linguistics. `https://doi.org/10.18653/v1/W19-4828`

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Sarkar Snigdha Sarathi Das, Ranran Haoran Zhang, Peng Shi, Wenpeng Yin, and Rui Zhang. 2023. Unified low-resource sequence labeling by sample-aware dynamic sparse finetuning. *arXiv preprint arXiv:2311.03748*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

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. https://doi.org/10.3115/1654536.1654538

Demi Guo, Alexander M. Rush, and Yoon Kim. 2020. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv: 2012.07463*. https://doi.org/10.18653/v1/2021.acl-long.378

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. 2024. LoRA+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*.

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*.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. *Advances in Neural Information Processing Systems*, 28.

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. https://doi.org/10.3115/v1/D14-1058

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*.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3651–3657, Florence, Italy. Association for Computational Linguistics. https://doi.org/10.18653/v1/P19-1356

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. https://doi.org/10.1162/tacl_a_00160

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, San Diego, California. Association for Computational Linguistics. https://doi.org/10.18653/v1/N16-1136

Neal Lawton, Anoop Kumar, Govind Thattai, Aram Galstyan, and Greg Ver Steeg. 2023. Neural architecture search for parameter-efficient fine-tuning of large pre-trained language models. *arXiv preprint arXiv:2305.16597*. https://doi.org/10.18653/v1/2023.findings-acl.539

Tao Lei, Junwen Bai, Siddhartha Brahma, Joshua Ainslie, Kenton Lee, Yanqi Zhou, Nan Du, Vincent Y. Zhao, Yuexin Wu, Bo Li, Yu Zhang, and Ming-Wei Chang. 2023. Conditional Adapters: Parameter-efficient transfer learning with fast inference. *Advances in Neural Information Processing Systems*, 36:8152–8172.

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*. https://doi.org/10.18653/v1/2023.acl-long.233

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. https://doi.org/10.18653/v1/P17-1015

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.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. DoRA: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*.

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. https://doi.org/10.1109/CVPR.2019.01152

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gulçehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence RNNs and beyond. *arXiv preprint arXiv:1602.06023*. https://doi.org/10.18653/v1/K16-1028

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*. https://doi.org/10.18653/v1/2021.naacl-main.168

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*. https://doi.org/10.18653/v1/2021.eacl-main.39

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. https://doi.org/10.18653/v1/D16-1264

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. https://doi.org/10.18653/v1/D15-1202

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

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. https://doi.org/10.18653/v1/D13-1170

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. https://doi.org/10.3115/1119176.1119195

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. 2023. DyLoRA: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 3274–3287. https://doi.org/10.18653/v1/2023.eacl-main.239

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*. https://doi.org/10.18653/v1/W18-5446

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641. https://doi.org/10.1162/tacl_a_00290

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. https://doi.org/10.18653/v1/N18-1101

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. https://doi.org/10.18653/v1/2020.emnlp-demos.6

Adam X.Yang, Maxime Robeyns, Xi Wang, and Laurence Aitchison. 2023. Bayesian low-rank adaptation for large language models. *arXiv preprint arXiv:2308.13111*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

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*. https://doi.org/10.18653/v1/2022.acl-short.1

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*.

Mingyang Zhang, Hao Chen, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, and Bohan Zhuang. 2023b. LoRAPrune: Pruning meets low-rank parameter-efficient fine-tuning. *arXiv preprint arXiv:2305.18403*. https://doi.org/10.18653/v1/2024.findings-acl.178

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023c. 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

Natural Language Understanding

For NLU, we evaluate all methods using the following eight tasks from the GLUE benchmark:

- 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.
- MRPC (Microsoft Research Paraphrase Corpus) (Dolan and Brockett, 2005): The goal is to determine semantic equivalency between the two input sentences.
- CoLA (Corpus of Linguistic Acceptability) (Warstadt et al., 2019): The task is to predict whether the given sentence is linguistically acceptable.
- STS-B (Semantic Textual Similarity Benchmark) (Cer et al., 2017): The task is to predict similarity of the given two sentences on a scale of 1 to 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.
- 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.
- QQP (Quora Question Pairs) (Wang et al., 2018): The task is to determine whether the questions in the given pair are semantically equivalent.
- MNLI (Multi-Genre Natural Language Inference) (Williams et al., 2018): The task is to determine the relationship between a given premise and hypothesis by predicting whether the premise entails the hypothesis, contradicts it, or neither. This dataset has two validation sets: matched (in-domain) and mismatched (cross-domain) data.

Token Classification

For token classification, we use the shared task of CoNLL-2003 (Tjong Kim Sang and De

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	10K	10K
NER	CoNLL-2003	14K	3.2K	3.5K
Summarization	CNN/Daily Mail	287113	13368	11490
Math Reasoning	Math10K	10K	–	–
	GSM8K	8.8K	–	1319
	SVAMP	–	–	1000
	MultiArith	–	–	600
	AddSub	–	–	395
	AQuA	100K	–	254
	SingleEq	–	–	508

Table 10: Datasets and data splits for different tasks used in the paper.

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.

Summarization

For summarization, we use the CNN/Daily Mail dataset (See et al., 2017), which consists of 300K unique news articles and their highlights written by journalists at CNN and the Daily Mail.

Generative Reasoning

For math reasoning tasks, we fine-tune the model using the Math10K dataset and evaluate the final model on the test-split of the following six datasets:

- 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 obtain the final answer.
- SVAMP (Patel et al., 2021): This dataset is created by introducing straightforward variations to single-unknown arithmetic word problems designed for grade levels up to 4.
- MultiArith (Roy and Roth, 2015): This dataset consists of multi-step arithmetic word problems involving basic operations, such as addition followed by subtraction or subtraction followed by division.
- AddSub (Hosseini et al., 2014): This corpus contains arithmetic problems with addition and subtraction.
- AQuA (Ling et al., 2017): This dataset contains algebraic word problems along with answer rationales.
- SingleEq (Koncel-Kedziorski et al., 2015): This dataset contains sentences expressing mathematical relations that form a single equation.
- Math10K (Hu et al., 2023): This dataset was constructed by combining training examples from GSM8K, AQuA, MAWPS and MAWPS-single (Koncel-Kedziorski et al., 2016). The original datasets contained only equations and final answers. To enhance them with explanations, the authors employed ChatGPT to generate reasoning steps for each example, creating the final Math10K dataset.

The train, validation and test splits of all the datasets are shown in Table 10.

8.2 Hyperparameters

All the common and task-specific hyperparameters are shown in Table 11a and 11b, respectively.

Common Hyperparameters

Budget For NLU and NER tasks, we use parameter budgets of 103K and 320K. The 103K budget is selected to align with the number of parameters fine-tuned using BitFit, which updates only the model's bias terms. The 320K budget is chosen to reduce the performance gap between PEFT methods and full fine-tuning. For summarization tasks, we adopt budgets of 100K, 320K, and 1M. These choices align with the NLU task

	Category	NLU	NER	Summarization	Math Reasoning
PEFT Method	hyperparameter	All tasks	CoNLL-2003	CNN/Daily Mail	All tasks
All methods	batch size	16	16	64	4
	learning rate	1×10^{-4}	1×10^{-4}	1×10^{-4}	3×10^{-4}
		3×10^{-4}	3×10^{-4}		
		5×10^{-4}	5×10^{-4}		
		7×10^{-4}	7×10^{-4}		
	seed	$\{6, 7, 8, 9\}$	$\{6, 7, 8, 9\}$	9	42
FFT	learning rate	5×10^{-6}	5×10^{-6}	1×10^{-5}	–
		7×10^{-6}	7×10^{-6}		
		1×10^{-5}	1×10^{-5}		
		3×10^{-5}	3×10^{-5}		
ID^3	exp	2	2	1	$\{0, 1\}$
	ϵ	1	1	1×10^{-3}	1
Fish	num_samples	1024	1024	–	–
	sample_type	``label''	``label''		
	grad_type	``square''	``square''		
LoRA	lora_r	8	–	–	$\{2, 8, 32\}$
	lora_alpha	8			$\{16, 64\}$
	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 hyperparameters.

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	CoNLL-2003	F1	20	300	384
Summarization	CNN/Daily Mail	Rouge-1/Rouge-2/Rouge-L	3	1000	Source length = 512 Target length = 128
Math Reasoning	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 hyperparameters.

Table 11: Details of all the hyperparameters used in the paper.

budgets while also including a larger budget for comparative analysis. For the mathematical reasoning task, the parameter budget for each model is set to match the number of parameters associated with applying LoRA at a rank of 2 to that model.

Learning Rate For NLU and NER tasks, we use learning rates of around 3×10^{-4} for selective fine-tuning. For full fine-tuning however, these learning rates are typically too high, and hence we use learning rates of around 7×10^{-6}. These learning rates were selected without bias toward any specific method and following the common practice of choosing rates within the range of 1×10^{-3} to 1×10^{-4}. For summarization and reasoning tasks, we fine-tune the models with learning rates of 1×10^{-4} and 3×10^{-4}, respectively.

Scoring For NLU and NER tasks, we conducted a single run for each learning rate, resulting in four runs per method. For each run, the maximum score based on the evaluation metric (accuracy or correlation) was recorded. The final score was calculated as the average of the four scores.

Specific Hyperparameters

PEFT-related For ID^3 we demonstrated in Section 6.2 that better results were achieved with positive values of exp. The parameter ϵ acts as a smoothing factor, with smaller values

Budget	Method	MNLI-m	MNLI-mm	QQP	QNLI	SST-2	STS-B	CoLA	MRPC	RTE	Avg
184M	Full-FT	0.30	0.40	0.36	0.29	0.36	0.50	1.67	1.00	1.83	0.74
103K	R-Mask	4.89	4.48	2.03	3.05	1.74	7.49	3.39	6.25	8.58	4.66
	Fish	0.48	0.51	0.29	0.43	0.33	0.79	2.17	1.10	2.90	1.00
	PaFi	0.82	0.62	0.73	0.89	0.62	1.80	1.80	1.80	2.51	1.29
	BitFit	1.08	0.75	0.70	0.89	0.51	1.67	1.09	1.42	2.91	1.23
	$\texttt{ID}^3$	0.12	0.13	0.27	0.17	0.25	0.27	0.45	0.42	3.14	0.58
320K	R-Mask	2.00	1.58	1.27	1.96	1.27	5.33	2.82	8.88	5.13	3.36
	Fish	0.12	0.11	0.45	0.13	0.16	0.36	1.39	0.47	2.45	0.63
	PaFi	0.92	0.92	0.79	0.46	0.60	1.07	1.41	0.82	2.57	1.06
	$\texttt{ID}^3$	0.29	0.38	0.27	0.11	0.11	0.28	1.45	0.37	1.60	0.54

(a) Standard deviations corresponding to the results in Table 1.

Budget	Method	MNLI-m	MNLI-mm	QQP	QNLI	SST-2	STS-B	CoLA	MRPC	RTE	Avg
1.33M	LoRA ($r = 8$)	0.23	0.12	0.12	0.36	0.21	0.29	1.42	1.32	0.86	0.55
320K	PaFi + LoRA ($r = 8$)	0.36	0.48	0.86	0.45	0.09	0.60	0.74	0.70	1.77	0.67
320K	$\texttt{ID}^3$ + LoRA ($r = 8$)	0.35	0.28	0.26	0.08	0.15	0.22	0.66	0.42	1.33	0.42

(b) Standard deviations corresponding to the results in Table 2.

Budget	Full-FT	Fish	PaFi	BitFit	$\texttt{ID}^3$	R-Mask
103K	–	0.41	1.66	1.69	0.64	31.09
320K	–	0.16	1.25	–	0.28	7.70
184M	0.20				–	

(c) Standard deviations corresponding to the results in Table 4.

Table 12: Standard deviations corresponding to GLUE and NER results.

generally yielding improved outcomes. As outlined in the Fish Mask paper (Sung et al., 2021), the optimal hyperparameters for ''num_samples,'' ''sample_type,'' and ''grad_type'' were used. Meanwhile, BitFit and PaFi do not use any hyperparameters.

Task-related The number of epochs, evaluation steps, and maximum sequence length are determined based on the size of the training and evaluation datasets. These details are presented in Table 11b.

8.3 Standard Deviations

We report the standard deviations on GLUE and NER tasks for all the baselines in Table 12.

8.4 Number of Runs

We report the number of runs (and the different hyperparameters used in these runs) conducted for each experiment in Table 13.

8.5 Statistical Significance Testing

We perform statistical tests for all the results reported in the paper. In particular, we perform the Wilcoxon signed-rank test,[3] a non-parametric test, since the nature of the accuracy distribution with different seeds and learning rates is not known. We perform the paired variant of these tests with data-points in a pair being drawn from the same configuration (task, budget). Table 14 provides for each result the configurations across which the test has been done. Further, in all cases where multiple runs for a given configuration are available (Tables 1, 2, and 4) we use what we call ''bootstrapping,'' where we pair each run of $\texttt{ID}^3$ with every run of the compared baseline, obtaining n^2 pairs (where n is the number of runs for a given configuration for each method). For instance, in the GLUE table (Table 1) we have two budgets (103K and 320K) and four runs per entry (obtained by varying the learning rate). So, in total we have 4×4 pairs for a given budget and in total $2 \times 4 \times 4 = 32$ pairs for a task (across the two budgets). As per common practice, we take the significance level as 0.05, which means that a result is considered statistically significant if its p-value < 0.05. For each result table, we take the best task-wise baseline (excluding full fine-tuning or its alternatives) as the paired method.

[3]Note that in order to use this test, the distribution must be symmetric about a center. We assume that this holds for all tasks.

Table/Figure	Number of runs for each method	Hyperparameter varied
Figure 1	4	Learning rate
Table 1	4	Learning rate
Table 2	4	Learning rate
Table 3	1	–
Table 4	4	Learning rate
Table 5	1	–
Table 6	1	–
Table 7	1	–
Figure 4	1	–
Figure 5	1	–
Figure 6a	4	Exponent
Figure 6b	4	Epsilon
Figure 7	1	–
Figure 8	1	–

Table 13: Number of runs and tuned hyperparameters for different experiments.

Table	**T**ested across	**B**ootstrapped	**P**aired method (Y)	**P**-value	**S**ignificant
Table 1	Budget	Yes	Task-wise best baseline	8×10^{-7}	Yes
Table 2	Task	Yes	PaFi + LoRA	0.04	Yes
Table 3	Task	–	SparseAdapter	0.12	No
Table 4	Budget	Yes	Fish Mask	0.01	Yes
Table 5	Budget	–	PaFi	0.13	No
Table 6	Model	–	Task-wise best baseline	0.06	No
Table 7	Task, budget	–	PaFi	0.01	Yes

Table 14: Description of all statistical significance tests. For all the tests we use the **null hypothesis:** *The observation X_i (ID^3) – Y_i (mentioned in column 3) is symmetric about $\mu = 0$.* We use an **alternative hypothesis:** *The observations X_i – Y_i are symmetric about $\mu > 0$.* The **significance level** is set as 0.05. For Tables 1, 2 and 4 (where results with multiple hyperparameter configurations are available), we use ''bootstrapping,'' where we compare all the pairwise results obtained by ID^3 and the best baseline for a given configurations.