Layer-wise Importance Matters: Less Memory for Better Performance in Parameter-efficient Fine-tuning of Large Language Models

Kai Yao[1,2*], **Penlei Gao**[3*], **Lichun Li**[2], **Yuan Zhao**[2],
Xiaofeng Wang[3], **Wei Wang**[2†], **Jianke Zhu**[1†],
[1]Zhejiang University [2]Ant Group [3]Cleveland Clinic Lerner Research Institution
jiumo.yk@antgroup.com, gaop@ccf.org

Abstract

Parameter-Efficient Fine-Tuning (PEFT) methods have gained significant popularity for adapting pre-trained Large Language Models (LLMs) to downstream tasks, primarily due to their potential to significantly reduce memory and computational overheads. However, a common limitation in most PEFT approaches is their application of a uniform architectural design across all layers. This uniformity involves identical trainable modules and ignores the varying importance of each layer, leading to sub-optimal fine-tuning results. To overcome the above limitation and obtain better performance, we develop a novel approach, Importance-aware Sparse Tuning (IST), to fully utilize the inherent sparsity and select the most important subset of full layers with effective layer-wise importance scoring. The proposed IST is a versatile and plug-and-play technique compatible with various PEFT methods that operate on a per-layer basis. By leveraging the estimated importance scores, IST dynamically updates these selected layers in PEFT modules, leading to reduced memory demands. We further provide theoretical proof of convergence and empirical evidence of superior performance to demonstrate the advantages of IST over uniform updating strategies. Extensive experiments on a range of LLMs, PEFTs, and downstream tasks substantiate the effectiveness of our proposed method, showcasing IST's capacity to enhance existing layer-based PEFT methods. Our code is available at https://github.com/Kaiseem/IST

1 Introduction

Significant achievements in natural language processing (NLP) have been achieved this year from the use of large language models (LLMs) that are pre-trained on extensive general datasets (Zhuang et al., 2024; Brown et al., 2020). These LLMs typically require full fine-tuning (FFT) (Howard and Ruder, 2018) to adapt them for specialized downstream tasks, an approach that necessitates retraining all model parameters. Nevertheless, as the size of these models and the volume of data increase, FFT becomes increasingly costly and impractical. Aiming to reduce the cost, parameter-efficient fine-tuning (PEFT) methods, involving adapter-based (Houlsby et al., 2019; Wang et al., 2022; Lei et al., 2024; He et al., 2022a), reparameterization-based (Hu et al., 2021; Edalati et al., 2022; Liu et al., 2024), and prompt-based methods (Li and Liang, 2021; Liu et al., 2022; Lester et al., 2021), have been proposed to reduce the number of trainable parameters in fine-tuning for the downstream tasks. However, most existing PEFT methods employ a uniform approach that indiscriminately assigns trainable parameters to identical positions across all layers, which could be unnecessary. This strategy relies heavily on human heuristics and overlooks task-specific domain gaps and characteristics, limiting their performance across various downstream tasks. Although some PEFT methods have improved the efficiency of fine-tuning LLMs, such as dynamic rank (Zhang et al., 2023b, 2024), they are tailored specifically for LoRA-based mod-

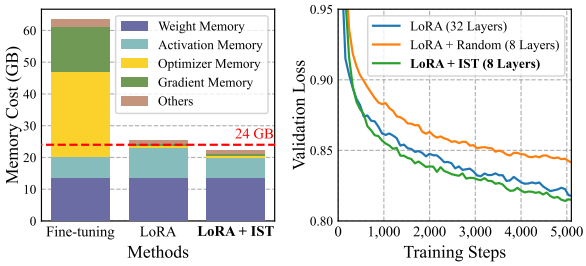


Figure 1: (Left) Memory consumption of tuning a LLaMA 7B model with a token batch size of 1024 on a single device. Details refer to Sec. 4.1. (Right) In comparison to the vanilla tuning of all 32 and random tuning of 8 LoRA layers, IST achieves a better validation loss.

*These authors contributed equally to this work.
†Corresponding authors.

Findings of the Association for Computational Linguistics: EMNLP 2024, pages 1977–1992
November 12-16, 2024

els and do not extend their benefits to the additional learnable module-based methods, i.e., Series and Parallel configurations. This limitation creates a clear necessity for a more generalized algorithm to enhance model performance across various domains.

Inspired by LISA (Pan et al., 2024), we empirically found that training a small fraction of the full layers in PEFT can yield comparably promising results to those achieved with FFT. The existing PEFT methods exhibit markedly redundancy in layer updating during the training process, leading us to investigate the differences among the layers of varying importance from the perspective of layer-wise sparsity. Motivated by these inherent insights, we propose a novel PEFT-compatible plug-and-play approach, **I**mportance-aware **S**parse **T**uning (IST), that estimates the task-specific importance score of each layer and fine-tunes the most important ones. As shown in Figure 1, our method substantially lowers memory demands during training by reducing the number of layers that require updates. Furthermore, by integrating layer-wise sparsity into our methodology, we enhance the convergence of layer-based PEFT methods, thereby achieving improved performance. The experimental results show that IST consistently improves existing layer-wise PEFT methods without sacrificing performance and inference efficiency across a wide range of models.

In summary, our contributions are as follows:

- Based on the empirical insight that sparse patterns markedly enhance the convergence of PEFT models, we propose an importance-aware sparse tuning method that prioritizes the most important layers for updating, making PEFT memory efficient and more powerful.
- We provide theoretical proof of convergence for the IST approach and present empirical evidence showing that it outperforms traditional uniform update strategies for PEFT.
- Extensive experiments in various LLMs, PEFT methods, and downstream tasks demonstrate the effectiveness and capacity of IST to enhance existing PEFT without sacrificing performance.

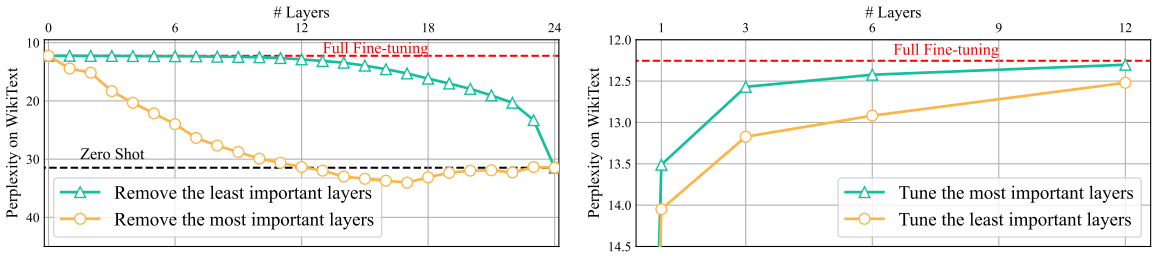
2 Related Work

2.1 Parameter-efficient Fine-tuning

As models grow in size and complexity, pre-trained Large Language Models (LLMs) have shown impressive performance across a range of natural language processing (NLP) tasks. However, efficiently adapting these LLMs to specific downstream tasks poses increasing challenges. Parameter-efficient fine-tuning (PEFT) addresses this dilemma by fine-tuning a few additional parameters or a subset of the pre-trained parameters. The existing PEFT approaches can be roughly categorized into three main types: adapter-based (Houlsby et al., 2019; Wang et al., 2022; Lei et al., 2024; He et al., 2022a), reparameterization-based (Hu et al., 2021; Edalati et al., 2022; Liu et al., 2024), and prompt-based methods (Li and Liang, 2021; Liu et al., 2022; Lester et al., 2021). Adapter-based methods focus on adding extra tunable parameters by introducing new layers within the original model. For example, Series Adapters (Houlsby et al., 2019) incorporate linear modules in a sequential manner, whereas Parallel Adapters (He et al., 2022a) add learnable modules in parallel with the model's existing sub-layers. Meanwhile, reparameterization-based methods aim to reduce the total number of trainable parameters by employing low-rank representations. LoRA (Hu et al., 2021), a notably effective and popular method, breaks down the delta parameter matrix into two lower-rank matrices. Yet, most current PEFT methods apply a uniform architectural approach across all layers, utilizing the same trainable modules for each layer. In this study, we present a novel approach that dynamically tunes a subset of full layers through PEFT, significantly enhancing both training efficiency and the performance of the fine-tuned models.

2.2 Layer-wise Sparse Tuning

Many previous works have uncovered the phenomenon of layer redundancy in pre-trained models, as evidenced by methods such as LayerSkip (Elhoushi et al., 2024), LayerDrop (Sajjad et al., 2023), LayerSharing (Zhang et al., 2023a; Lan et al., 2020), and structured pruning (Fan et al., 2021; Zhang and He, 2020). This indicates that the importance of each layer could be different, and not all layers need fine-tuning. However, selecting the appropriate layers for fine-tuning downstream tasks remains a significant challenge. Lee et al. (2023) suggests selectively fine-tuning a subset of layers depending on the type of domain shift. Similarly, Kaplun et al. (2023) deploys a greedy search to find the most suitable layers for fine-tuning, demanding considerable computational resources and time for initiation. Recently, layer-wise sparse training for large language models has become a popular topic.



(a) Remove trained LoRA modules layer-by-layer greedily

(b) Train LoRA modules within the selective layers

Figure 2: Illustration of layer redundancy in PEFT training on the OPT-1.3B. (a) A greedy selection strategy is employed to iteratively remove the trained LoRA modules from the model. (b) Specific layers of the model are selectively fine-tuned using LoRA. The importance of layers depends on their contribution to the performance.

For example, LISA (Pan et al., 2024) randomly selects a subset of layers to be optimized during training, leading to promising faster convergence and better performance. LIFT (Zhu et al., 2024) selects one layer to fine-tune LLMs with different selection strategies such as front-to-end, end-to-front, or random, obtaining comparable performance while reducing the computational load. Although effective, these methods require substantial storage equivalent to the full model since all parameters are being updated. Furthermore, these approaches do not deeply explore joint use with PEFT and have employed relatively simple selection strategies, limiting their performance. Unlike these previous methods, we focus on integrating existing layer-based PEFT and propose an importance-aware layer selection strategy that significantly enhances performance while increasing efficiency.

3 Method

3.1 Motivation

To showcase the excessive layer redundancy in training PEFT, we conducted empirical evaluations on the OPT 1.3B (Zhang et al., 2023c) model fine-tuned on the WikiText (Merity et al., 2016) dataset. Initially, we adopted LoRA on all model's layers and trained it on this dataset. After training, we employed a greedy selection strategy to remove the least or the most important layers individually according to their contribution to the model's performance. On the one hand, as illustrated in Figure 2(a), removing 50% of the least important LoRA layers did not substantially elevate perplexity. On the other hand, removing the most important LoRA layers resulted in a rapid decline in performance. These preliminary findings indicate inherent layer-wise sparsity during PEFT training, leading to the phenomenon that not all layers are effectively trained with PEFT.

This observation prompts us to question: *what causes the layer-wise sparsity?* To answer this question, we utilized the outcome of the greedy search to rank the layers according to their contribution to the model's performance. Next, we performed PEFT fine-tuning on the most and least important layers. As shown in Figure 2(b), even when only a small portion of the layers (or merely a single one) are trained using LoRA, it is possible to attain comparable results to those obtained through full fine-tuning (FFT). This suggests that the observed sparsity is not due to the unimportant layers of the original network. Instead, it implies that the layer-wise sparsity observed in PEFT is an inherent characteristic, naturally emerging throughout the network's training process. Furthermore, training more important layers yields better outcomes consistently than focusing on the less important ones, emphasizing the beneficial role of importance in layer-wise sparsity.

3.2 Convergence of Layer-wise Sparse Tuning

In the following, we will demonstrate why layer-wise sparse tuning is efficient and effective during fine-tuning. In particular, we develop proof that if we randomly select a subset of full layers in the layer-based PEFT method and only update these selected parameters, the risk bond of the subsets can be tighter than updating the whole layers.

Given a pretrained Large Language Model (LLM) $\mathcal{M} = \{m_1, m_2, \ldots, m_{N_L}\}$, comprising N_L layers and parameterized by Θ, alongside a downstream dataset $\mathcal{D} = \{(x_i, y_i)\}_{i \in [|\mathcal{D}|]}$, full fine-tuning (FFT) this model on the downstream dataset achieve $\mathcal{M}_\Theta \rightarrow \mathcal{M}_{\Theta+\Delta}$, $\Delta = \arg\min_\Delta \mathcal{L}(\Theta + \Delta, \mathcal{D})$. PEFT introduces a learnable module $\mathcal{A}$ with a significantly smaller number of trainable parameters, denoted as $\mathcal{M}' = [\mathcal{M}_\Theta, \mathcal{A}]$, where $|\theta^{\mathcal{A}}| \ll |\Delta|$, aiming to achieve performance com-

parable to the fully fine-tuned model $\mathcal{M}_{\Theta+\Delta}$. The empirical loss over the training set $\mathcal{D}$ is defined as $\mathcal{L}(\theta^{\mathcal{A}};(x,y)) = \frac{1}{|D|}\sum_{i\in[|\mathcal{D}|]}\ell(y_i, f(x_i;\theta^{\mathcal{A}}))$, where ℓ denotes a suitable loss function, such as cross-entropy.

The learnable module $\mathcal{A}$ in most existing PEFT methods can be represented as $\mathcal{A} = \{a_1, a_2, \ldots, a_{N_L}\}$. According to the sparse tuning strategy, the total layers of the given adapter module can be divided into two groups: S and $\bar{S}$ represent a set of randomly selected layers that are updated and a set of remaining layers that are frozen during fine-tuning respectively. The total parameter vector $\theta^{\mathcal{A}}$ is then partitioned into $\theta^{\mathcal{A}}_S$ and $\theta^{\mathcal{A}}_{\bar{S}}$. The loss function can conceptually be decomposed as follows:

$$\mathcal{L}(\theta^{\mathcal{A}};(x,y)) = \mathcal{L}(\theta^{\mathcal{A}}_S, \theta^{\mathcal{A}}_{\bar{S}};(x,y)). \quad (1)$$

Corollary 3.1 *Consider the Taylor expansion of the loss function $\mathcal{L}(\theta^{\mathcal{A}}_S, \theta^{\mathcal{A}}_{\bar{S}})$ around $\theta^{\mathcal{A}}_{\bar{S}}$:*

$$\begin{aligned}\mathcal{L}(\theta^{\mathcal{A}}_S, \theta^{\mathcal{A}}_{\bar{S}}) &= \mathcal{L}(\theta^{\mathcal{A}}_S, \theta^{\mathcal{A}}_{\bar{S}0}) \\ &+ \nabla_{\theta^{\mathcal{A}}_{\bar{S}}}\mathcal{L}(\theta^{\mathcal{A}}_S, \theta^{\mathcal{A}}_{\bar{S}0})\top(\theta^{\mathcal{A}}_{\bar{S}} - \theta^{\mathcal{A}}_{\bar{S}0}) \\ &+ \mathcal{O}((\theta^{\mathcal{A}}_S)^2),\end{aligned} \quad (2)$$

where $\theta^{\mathcal{A}}_{\bar{S}0}$ represents the fixed parameters before any fine-tuning. Since $\theta^{\mathcal{A}}_{\bar{S}}$ does not change during fine-tuning process, we can set $\theta^{\mathcal{A}}_{\bar{S}} = \theta^{\mathcal{A}}_{\bar{S}0}$. The first-order term of Eq. 2 can be eliminated and we can have the approximate loss:

$$\mathcal{L}(\theta^{\mathcal{A}}_S, \theta^{\mathcal{A}}_{\bar{S}}) \approx \mathcal{L}(\theta^{\mathcal{A}}_S, \theta^{\mathcal{A}}_{\bar{S}0}) \propto \mathcal{L}(\theta^{\mathcal{A}}_S). \quad (3)$$

This estimation shows that the loss function mainly depends on the updates of $\theta^{\mathcal{A}}_S$, supporting the decision to focus updates on the subset of full layers.

In Vapnik–Chervonenkis (VC) theory (Devroye et al., 1996), the VC-dimension denoted as $VCdim(\mathcal{H})$ is a measure of the size, i.e., capacity, complexity, expressive power, richness, or flexibility, of a class of sets $\mathcal{H}$. For neural networks, including LLMs, the VC-dimension typically increases with the number of trainable parameters. Let d_S be the VC-dimension of subset $\mathcal{H}_S$ and d be the VC-dimension of full set $\mathcal{H}$. By updating only a subset of parameters $\theta^{\mathcal{A}}_S$, the effective VC-dimension d_S of the hypothesis class corresponding to these parameters is reduced, which leads to a tighter generalization bound:

Lemma 3.2 *With a probability at least $1-\delta$ over the choice of a training set of size n, the following bound holds for $\mathcal{H}_S \subseteq \mathcal{H}$:*

$$\begin{aligned}&|\mathcal{R}(\mathcal{H}) - \hat{\mathcal{R}}_n(\mathcal{H})| \approx |\mathcal{R}(\mathcal{H}_S) - \hat{\mathcal{R}}_n(\mathcal{H}_S)| \\ &\leq \sqrt{\frac{Cd_S log(n/d_S) + log(1/\delta)}{n}},\end{aligned} \quad (4)$$

where $\mathcal{R}(\mathcal{H}_S) = \mathbb{E}_{(x,y)\sim D}\mathcal{L}(\theta^{\mathcal{A}}_S; x, y)$ denotes the expected risk under the data distribution D and $\hat{\mathcal{R}}_n(\mathcal{H}_S) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(\theta^{\mathcal{A}}_S; x_i, y_i)$ denotes the generalization risk on the specific dataset. C is a constant related to the model and data distribution. Since $d_S \leq d$, the generalization bound becomes tighter, implying that models with fewer updating layers generalize better assuming the same number of training samples.

Based on Eq. 3, the generalization error of the model can be formally estimated as $|\mathcal{R}(\mathcal{H}) - \hat{\mathcal{R}}_n(\mathcal{H})| \approx |\mathcal{R}(\mathcal{H}_S) - \hat{\mathcal{R}}_n(\mathcal{H}_S)|$.

When $\theta^{\mathcal{A}}_S$ is updated and $\theta^{\mathcal{A}}_{\bar{S}}$ remains fixed, the model effectively reduces the dimensionality of the optimization problem. This can potentially lead to a more focused and efficient parameter search:

Corollary 3.3 *The derivative of $\mathcal{L}(\theta^{\mathcal{A}})$ with respect to $\theta^{\mathcal{A}}_S$ can be obtained as:*

$$\frac{\partial\mathcal{L}(\theta^{\mathcal{A}})}{\partial\theta^{\mathcal{A}}_S} = \frac{1}{|D|}\sum_{i\in[|\mathcal{D}|]}\frac{\partial\ell((y_i, f(x_i;\theta^{\mathcal{A}})))}{\partial\theta^{\mathcal{A}}_S}. \quad (5)$$

The magnitude and direction of this gradient tell us how sensitive the empirical risk is to changes in $\theta^{\mathcal{A}}_S$ and hence guide the updates during training.

From the above analysis, we found that noticeable patterns of sparsity combined with the smoothness of the objective function, can markedly improve the rate of convergence, potentially leading to a linear speed-up. To achieve improved error bounds and convergence rates, the crucial strategy lies in selecting the most important layers of the full model that are particularly pertinent to the specific task. This selection process involves identifying which layers contribute the most to task-specific performance, enabling a more focused and efficient training regimen.

3.3 Importance-aware Sparse Tuning

In the previous section, we proved that sparse tuning leads to a better convergence for downstream task fine-tuning. In this section, we will introduce our method, Importance-aware Sparse Tuning (IST), aiming to enhance the performance of layer-wise sparse tuning motivated by empirical

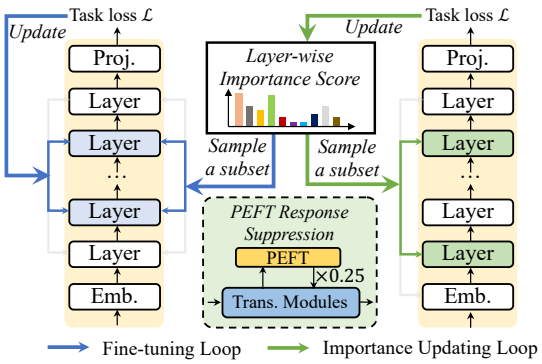


Figure 3: Workflow of Importance-aware Sparse Tuning (IST): IST consists of two main loops: a fine-tuning loop, which selects a subset of layers for updating PEFT modules, and an importance updating loop, which estimates layer-wise importance by assessing the response suppression of the selected PEFT modules.

observations. IST involves two loops: the fine-tuning loop, which selects a subset of full layers to update the PEFT modules, and the importance updating loop, which updates the importance score of each layer. To estimate layer-wise importance more accurately, we dynamically select the subsets of all layers for PEFT response suppression during the importance updating process. Drawing inspiration from reinforcement learning, which explores the best structure based on rewards (Zoph and Le, 2017; Pham et al., 2018; Liu et al., 2017), we treat the layer selection process as a multi-armed bandit problem and use reinforcement learning to obtain the importance score of each layer.

Fine-tuning Loop Formally, given a PEFT-equipped LLM for downstream data fine-tuning, $\mathcal{M}' = [\mathcal{M}, \mathcal{A}] = \{m_i, a_i\}_{i=1}^{N_L}$, where m_i is frozen and a_i is trainable, our goal is to generate a subset S of full layers to update, and keep the remaining set $\bar{S}$ unchanged. To this end, we first define the degree of importance as $\mathbf{I} \in \mathbb{R}^{N_L}$, which is zero-initialized and updated through fine-tuning process simultaneously. In each training iteration, we choose N_u layers to update based on $\mathbf{I}$. For t-th step, the action policy π_i for i-th layer follows the uniform distribution:

$$\pi_i \sim U(0, \text{Sigmoid}(\mathbf{I}_i)). \tag{6}$$

We randomly sample probability score p_i for each layer, i.e., $p_i \sim \pi_i$. The subset S can be determined with the score p_i:

$$S = \{i | p_i > p_{N_u}\}, \bar{S} = \{i | p_i \leq p_{N_u}\}, \tag{7}$$

where p_{N_u} is the N_u largest values in the sampled probabilities. Then, the chosen PEFT modules are updated by $\theta_{a_i, i \in S} \leftarrow \nabla_{\theta^{\mathcal{A}}} \mathcal{L}(\theta^{\mathcal{A}})$

Importance Updating Loop To update the importance score, we suppress the response of a_i to measure its contribution to the result. If a_i is relatively important, reducing its response will significantly increase the loss, and vice versa. We sample N_c candidate sets $\{S_c^1, \ldots, S_c^{N_c}\}$, each containing N_v layers. For the j-th sampling, we reduce the response of a_i for the layers that were not selected:

$$o_{i+1}^j = \begin{cases} m_i(o_i^j) + a_i(o_i^j) & \text{if } i \in S_c^j \\ m_i(o_i^j) + \beta * a_i(o_i^j) & \text{otherwise} \end{cases}, \tag{8}$$

where $\beta \in [0, 1]$ is the response suppression factor. Then, for the j-th sampled set S_c^j, we calculate the rewards according to their loss:

$$\mathbf{r}^j = e^{-\mathcal{L}^j} - \frac{1}{N_c} \sum_{k=1}^{N_c} e^{-\mathcal{L}_k}. \tag{9}$$

Due to the smaller contributions of PEFT compared to the original network, the response suppression of PEFT may lead to relatively small reward values. Therefore, we employed a large updating rate μ to accelerate the convergence of importance, ensuring it matches the fine-tuning process:

$$\mathbf{I}_i = \begin{cases} \mathbf{I}_i + \mu * \mathbf{r}_j & \text{if } i \in S_c^j \\ \mathbf{I}_i & \text{otherwise} \end{cases}, \tag{10}$$

where μ controls the convergence of importance.

Joint Training We propose jointly training IST with PEFT to avoid the costly greedy search observed in prior studies (Kaplun et al., 2023), as shown in Figure 3. Specifically, to align with the training dynamics of PEFT, we execute the importance updating loop every T_c fine-tuning loop. While our method reduces the time required for the fine-tuning loop slightly, it introduces additional forward time within the importance updating loop. Consequently, we set $T_c = 10$ and $N_c = 3$ to keep the training time efficient.

4 Experimental Results

In this section, we conduct a series of experiments to validate the effectiveness of our proposed IST. We integrate IST into the Series Adapter, Parallel Adapter, and LoRA, and then compare them with their original counterparts across various tasks.

Model	Weight Mem.	Full Fint-tuning	PEFT			PEFT + IST		
			Series	Parallel	LoRA	Series	Parallel	LoRA
GPT2-Small$_{120M}$	0.4G	3.4G	2.5G	2.8G	2.9G	2.0G	2.1G	2.2G
TinyLLaMA$_{1.1B}$	2.2G	15.9G	9.6G	10.3G	10.5G	8.0G	8.3G	8.5G
LLaMA$_{7B}$	14G	60G	22G	23G	24G	19G	20G	20G
LLaMA$_{13B}$	27G	OOM	38G	41G	42G	35G	36G	36G

Table 1: Comparison of memory consumption for various LLMs and PEFT methods.

Baselines We included the following widely used layer-based fine-tuning methods.

- **Full Fine-tuning** (Howard and Ruder, 2018) - All parameters within the pre-trained model are optimized during training.
- **Series Adapter** (Houlsby et al., 2019) - Additional learnable modules are introduced into a specific sublayer in a sequential manner.
- **Parallel Adapter** (He et al., 2022a) - Additional learnable modules are integrated in parallel with distinct sublayers within the backbone model.
- **LoRA** (Hu et al., 2021) - Parameter efficiency is enhanced by decomposing the learnable delta parameter matrix into two low-rank matrices.

For the optimal configuration and placement of PEFT methods, we adhere to the settings established by Hu et al. (2023). Specifically, Series and Parallel Adapters are seamlessly integrated into the MLP layers with a bottleneck size of 256. Similarly, LoRA is seamlessly incorporated into both the Multi-head Self-attention layers and the MLP layers, with a rank of 32. Across all PEFT methods, we maintain the same tunable parameter budgets, adjusting only the learning rate. For IST, we consistently set N_u to 25% of the layers for the fine-tuning loop, N_v to 50% of the layers for the importance updating loop, and β to 0.25. Further details on the experimental settings are available in the Appendix.

4.1 Memory Efficiency

We conducted experiments on maximum GPU memory to demonstrate the efficiency of IST in terms of memory usage, revealing that it requires less memory compared to standard PEFT methods.

Settings To obtain an accurate estimation of the memory, we randomly sampled prompts from the Alpaca (Peng et al., 2023) dataset and restricted the maximum output token length to 1024. We uniformly employed a mini-batch size of 1 across four LLMs, ranging from 120M to 13B parameters, and three types of PEFT methods. We presented the overall memory consumption, consisting of weight memory, activation memory, optimizer memory, and gradient memory. Additionally, we separately demonstrated weight memory to highlight the significant role of IST in reducing training memory. To isolate the impact of the evaluated variables, we excluded GPU memory-saving techniques, such as gradient checkpointing (Chen et al., 2016), offloading (Ren et al., 2021), and flash attention (Dao et al., 2022).

Results We list the memory consumption for various LLMs and PEFT methods in Table 1. The overall results show that training LLMs with our proposed IST strategy could significantly reduce memory consumption in all the widely used LLMs compared to full fine-tuning and standalone PEFT configurations. Combining PEFT modules and our proposed IST could save a lot of training memory including activation memory, optimizer memory, and gradient memory on the three popular adapters. As for the LLaMA 7B model, training with IST could almost reduce the average 36% training memory for all the PEFT methods. This trend of reduced memory usage with IST integration is consistent across other models as well. These results highlight the effectiveness of IST in enhancing the memory efficiency of fine-tuning LLMs, which makes IST a valuable strategy in deploying more resource-efficient fine-tuning practices, especially important for scenarios where computational resources are a limiting factor.

4.2 Commonsense Reasoning

Settings To validate the effectiveness of IST, we evaluated three PEFT methods across five LLMs on the commonsense reasoning tasks. Specifically, the adaptability of PEFT was verified using Series, Parallel Adapter, and LoRA methods on the LLaMA 7/13B (Touvron et al., 2023) models, and the adaptability of LLM was tested on three models: GPT-J 6B (Wang and Komatsuzaki, 2021),

Model	PEFT	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
ChatGPT	-	73.1	85.4	68.5	78.5	66.1	89.8	79.9	74.8	77.0
LLaMA_{7B}	Series	63.0	79.2	76.3	67.9	75.7	74.5	57.1	72.4	70.8
	Series + IST	66.2	78.3	74.9	72.2	75.9	75.8	59.0	72.2	**71.8**
	Parallel	67.9	76.4	78.8	69.8	78.9	73.7	57.3	75.2	72.2
	Parallel + IST	68.4	79.1	77.9	70.0	78.9	81.2	62.3	77.6	**74.4**
	LoRA	68.9	80.7	77.4	78.1	78.8	77.8	61.3	74.8	74.7
	LoRA + IST	68.7	81.7	77.3	82.7	78.7	80.6	62.4	80.0	**76.5**
LLaMA_{13B}	Series	71.8	83.0	79.2	88.1	82.4	82.5	67.3	81.8	79.5
	Series + IST	72.9	82.2	81.4	87.9	84.0	82.7	69.1	81.1	**80.2**
	Parallel	72.5	84.9	79.8	92.1	84.7	84.2	71.2	82.4	**81.4**
	Parallel + IST	72.6	86.0	79.2	89.1	83.5	84.8	70.6	82.8	81.1
	LoRA	72.1	83.5	80.5	90.5	83.7	82.8	68.3	82.4	80.5
	LoRA + IST	71.5	85.0	81.2	89.1	84.2	84.0	70.1	81.8	**80.9**
GPT-J_{6B}	LoRA	62.4	68.6	49.5	43.1	57.3	43.4	31.0	46.6	50.2
	LoRA + IST	63.0	63.2	62.9	35.8	39.1	56.8	39.1	51.2	**51.4**
BLOOMz_{7B}	LoRA	65.9	75.3	74.5	57.3	72.5	74.6	57.8	73.4	**68.9**
	LoRA + IST	67.0	74.4	74.4	51.4	68.7	77.9	58.9	74.4	68.4
LLaMA3_{8B}	LoRA	70.8	85.2	79.9	91.7	84.3	84.2	71.2	79.0	80.8
	LoRA + IST	72.7	88.3	80.5	94.7	84.4	89.8	79.9	86.6	**84.6**

Table 2: Accuracy comparison of multiple LLMs with various PEFT methods on eight commonsense reasoning datasets. Results of all the baseline methods on GPT-J, BLOOMZ and LLaMA are taken from Hu et al. (2023).

BLOOMZ 7B (Muennighoff et al., 2022), and LLaMA3 8B (AI@Meta, 2024). We also report ChatGPT's accuracy obtained with gpt-3.5-turbo API using a zero-shot Chain of Thought (Wei et al., 2022). The commonsense reasoning tasks consisted of 8 sub-tasks, each with a predefined training and testing set, including BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC (Clark et al., 2018) and OBQA (Mihaylov et al., 2018). Aligning with the setting of Hu et al. (2023), we aggregated the training data from all eight tasks to form the training dataset and conducted evaluations on the individual testing dataset for each task.

Results The quantitative results in Table 2 offer a comprehensive view of the performance improvements brought by the proposed IST method across various LLMs and PEFT configurations. We can see that IST consistently shows an enhancement in model performance on the commonsense reasoning task. Analyzed from the LLaMA 7B model, IST shows significant performance gains across multiple tasks compared to its PEFT-only counterparts in all three PEFT configurations. Notably, in tasks HellaSwag and QBQA, there's a noticeable improvement, demonstrating how IST can refine the model's response to more complex queries. Moreover, the impact of IST is not limited to one model or configuration. For example, in GPT-J 6B and BLOOMZ 7B, the IST enhancements lead to better outcomes in almost all tasks compared to LoRA configurations without IST. This across-the-board improvement underscores IST's robustness and general applicability. IST's ability to focus on the most impactful layers makes the fine-tuning process not only more memory efficient but also strategically adaptable to various reasoning tasks. This is particularly beneficial in scenarios where model responsiveness and accuracy are critical. The aggregation of training data across different tasks and the subsequent application of IST likely helps in developing a more generalized understanding of commonsense reasoning, making IST a valuable addition to the PEFT techniques.

Method	GSM8K	AQuA	MAWPS	SVAMP	Avg.
ChatGPT	56.4	38.9	87.4	69.9	63.2
LoRA	61.0	26.4	91.6	74.4	63.4
LoRA + IST	62.8	31.5	89.9	76.3	**64.7**

Table 3: Accuracy comparison of LLaMA3 8B on four math reasoning datasets.

4.3 Arithmetic Reasoning

Settings To further demonstrate IST's scalability on different tasks, we conduct additional fine-tuning experiments on arithmetic reasoning. We utilized LoRA to fine-tune the LLaMA 3 8B model. Similarly, we included the results from ChatGPT

Method	# Layers	Results
Vanilla Tuning	32	74.7
Random Sparse Tuning	4	67.1 (-7.6)
Importance-aware Sparse Tuning	4	73.7 (-1.0)
Random Sparse Tuning	8	75.8 (+1.1)
Importance-aware Sparse Tuning	8	76.5 (+1.8)

Table 4: Ablation studies on key components of IST.

Method	LoRA	LISA	AdaLoRA	LoRA +IST	AdaLORA +IST
Results	74.7	75.3	76.2	76.5	77.1

Table 5: Comparison with other adaptive methods.

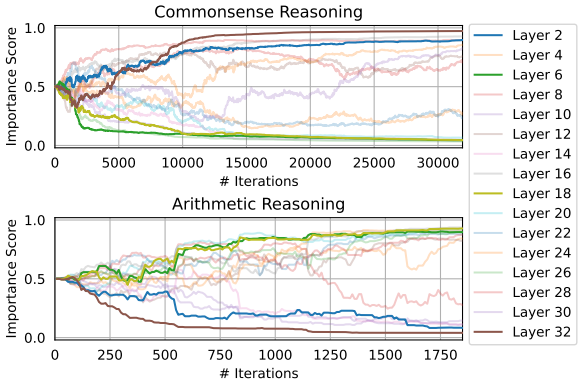


Figure 4: Layer-wise importance on different tasks.

3.5 as a reference, obtained using Zero-shot Chain-of-Thought (Wei et al., 2022). The fine-tuning process was conducted on the Math10K dataset, comprising math reasoning samples collected by Hu et al. (2023). Following the completion of training, we evaluated the model's performance on predefined test sets from several datasets, including GSM8K (Cobbe et al., 2021), AQuA (Ling et al., 2017), MAWPS (Koncel-Kedziorski et al., 2016), and SVAMP (Patel et al., 2021).

Results Table 3 shows the results of the arithmetic reasoning task. The accuracy on GSM8K and SVAMP datasets shows a consistent improvement from ChatGPT to LoRA, and further enhancement when IST is applied alongside LoRA, indicating the effectiveness of fine-tuning and IST in improving model performance for these datasets. The results of the AQuA dataset indicate a decrease in accuracy for LoRA compared to ChatGPT, but the application of IST helps to recover some of the lost performance. This suggests that while LoRA alone may not be as effective for AQuA, IST can mitigate some issues. Overall, combining IST and existing PEFT methods presents a robust approach for fine-tuning LLMs, leading to better generalization and accuracy in arithmetic reasoning tasks.

4.4 Analytical Study

Effect of Importance-aware Sparse Tuning We conducted experiments to evaluate the effects of importance-aware sparse tuning by training the LLaMA 7B model with LoRA on a commonsense task, reporting the average accuracy. As shown in Table 4, using sparse tuning with randomly selected layers, particularly with only four layers, does not yield satisfactory results. This outcome contrasts with findings from LISA (Pan et al., 2024) and LIFT (Zhu et al., 2024), where training very few layers (1-2 layers) resulted in a good performance. The discrepancy arises because, in LISA and LIFT, training entire transformer layers encompasses a substantial number of trainable parameters. Conversely, PEFT involves relatively fewer parameters, necessitating the fine-tuning of more layers to achieve better results. When we increased the number of sparse tuning layers to 8, we observed a considerable improvement of 1.1, aligning with theoretical expectations that sparse tuning enhances convergence. Finally, incorporating importance-aware tuning yielded the best results, underscoring the effectiveness of IST.

Comparison with Adaptive Methods We compared our proposed IST method with other adaptive methods, such as LISA (Pan et al., 2024) and AdaLoRA (Zhang et al., 2023b), with LLaMa 7B on the commonsense task to demonstrate our effectiveness. Notably, LISA is a PEFT method that focuses on sparsely tuning a single transformer layer, while AdaLoRA uses adaptive rank allocation and can widely adapt to reparameterization-based methods. As shown in the Table 5, compared to LoRA, LISA improved the average accuracy by 0.6, validating the concept of sparse training. AdaLoRA improved accuracy by 1.5, highlighting the importance of rank-level sparsity. Finally, our method can be combined with LoRA and AdaLoRA to further enhance performance, showcasing the broad applicability and practicality of IST.

Layer-wise Importance Learning We visualize the layer-wise importance of the two tasks with IST in Figure 4. The importance scores converge as the training iterations increase. The observed variation in the importance scores of each layer across different tasks indicates distinct levels of signifi-

cance. For instance, 'Layer 2' and 'Layer 32' significantly contribute to the commonsense reasoning task, whereas they are less important for the arithmetic reasoning task. Conversely, 'Layer 6' and 'Layer 18' exhibit contrasting importance levels across these tasks as well. This layer-wise differentiation underscores the effectiveness of our method, similar to curriculum learning, where the model progressively focuses on the most pertinent layers at each stage of training. By dynamically adjusting the importance of different layers, our approach allows for a more refined and task-specific tuning process, thereby enhancing the model's adaptability and performance across diverse tasks.

5 Conclusion

In this study, we proposed a novel Importance-aware Sparse Tuning (IST) approach for PEFT of LLMs. By dynamically selecting the most important layers in the fine-tuning loop, IST achieves a significant reduction in memory usage and computational overhead. The importance updating loop refines the selection of layers using a reinforcement learning approach, ensuring that the most impactful layers are prioritized during training. This innovative method leverages the inherent sparsity of layer-wise importance, leading to more efficient and effective fine-tuning through extensive experiments across various LLMs, PEFT methods, and downstream tasks. The proposed method holds promise for future applications where resource constraints and performance are critical considerations.

Acknowledgements

This work was supported by Ant Group Postdoctoral Programme.

Limitations

There are three limitations in this work. First, since IST employs reinforcement learning, tuning six related hyperparameters is required. Even after fixing three of these parameters, the search space for the remaining three remains large, possibly leading to increased trial-and-error costs during usage. Second, due to limited resources, we were unable to validate larger language models such as the LLaMA3 70B. These larger models exhibit stronger language comprehension capabilities and, consequently, yield better performance. Third, we did not thoroughly explore the variants or combinations of each PEFT method. Given the substantial computational demands and extensive hyperparameter search space, we leave this as future work.

References

AI@Meta. 2024. Llama 3 model card.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.

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.

Jiaao Chen, Aston Zhang, Xingjian Shi, Mu Li, Alex Smola, and Diyi Yang. 2023. Parameter-efficient fine-tuning design spaces. *arXiv preprint arXiv:2301.01821*.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*.

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

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

Luc Devroye, László Györfi, Gábor Lugosi, Luc Devroye, László Györfi, and Gábor Lugosi. 1996. Vapnik-chervonenkis theory. *A probabilistic theory of pattern recognition*, pages 187–213.

Ali Edalati, Marzieh S. Tahaei, Ivan Kobyzev, V. Nia, James J. Clark, and Mehdi Rezagholizadeh. 2022. Krona: Parameter efficient tuning with kronecker adapter. *ArXiv*, abs/2212.10650.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. 2024. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*.

Chun Fan, Jiwei Li, Xiang Ao, Fei Wu, Yuxian Meng, and Xiaofei Sun. 2021. Layer-wise model pruning based on mutual information. *arXiv preprint arXiv:2108.12594*.

Cheng Fu, Hanxian Huang, Xinyun Chen, Yuandong Tian, and Jishen Zhao. 2021. Learn-to-share: A hardware-friendly transfer learning framework exploiting computation and parameter sharing. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 3469–3479. PMLR.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. *ArXiv*, abs/2403.14608.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022a. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*.

Shwai He, Liang Ding, Daize Dong, Jeremy Zhang, and Dacheng Tao. 2022b. SparseAdapter: An easy approach for improving the parameter-efficiency of adapters. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2184–2190, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

James Henderson, Sebastian Ruder, et al. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. In *Advances in Neural Information Processing Systems*.

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.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

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, Yihuai Lan, Lei Wang, Wanyu Xu, Ee-Peng Lim, Roy Ka-Wei Lee, Lidong Bing, and Soujanya Poria. 2023. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Empirical Methods in Natural Language Processing*.

Gal Kaplun, Andrey Gurevich, Tal Swisa, Mazor David, Shai Shalev-Shwartz, and Eran Malach. 2023. Less is more: Selective layer finetuning with subtuning. *arXiv preprint arXiv:2302.06354*.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *Proceedings of NAACL*, pages 1152–1157.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representation*.

Yoonho Lee, Annie S Chen, Fahim Tajwar, Ananya Kumar, Huaxiu Yao, Percy Liang, and Chelsea Finn. 2023. Surgical fine-tuning improves adaptation to distribution shifts. In *International Conference on Learning Representation*.

Tao Lei, Junwen Bai, Siddhartha Brahma, Joshua Ainslie, Kenton Lee, Yanqi Zhou, Nan Du, Vincent Zhao, Yuexin Wu, Bo Li, et al. 2024. Conditional adapters: Parameter-efficient transfer learning with fast inference. *Advances in Neural Information Processing Systems*, 36.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Empirical Methods in Natural Language Processing*.

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

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.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2017. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*.

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. In *International Conference on Machine Learning*.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-tuning v2: Prompt tuning can be comparable to

fine-tuning universally across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association of Computational Linguistics*.

Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Wen tau Yih, and Madian Khabsa. 2021. Unipelt: A unified framework for parameter-efficient language model tuning. *ArXiv*, abs/2110.07577.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering.

Niklas Muennighoff, Thomas Wang, Lintang Sutawika, Adam Roberts, Stella Biderman, Teven Le Scao, M Saiful Bari, Sheng Shen, Zheng-Xin Yong, Hailey Schoelkopf, et al. 2022. Crosslingual generalization through multitask finetuning. *arXiv preprint arXiv:2211.01786*.

Rui Pan, Xiang Liu, Shizhe Diao, Renjie Pi, Jipeng Zhang, Chi Han, and Tong Zhang. 2024. Lisa: Layerwise importance sampling for memory-efficient large language model fine-tuning. *arXiv preprint arXiv:2403.17919*.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of NAACL*, pages 2080–2094.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning (ICML)*, pages 4092–4101.

Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. 2021. Zero-offload: Democratizing billion-scale model training. *arXiv preprint arXiv:2101.06840*.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2023. On the effect of dropping layers of pre-trained transformer models. *Computer Speech & Language*, 77:101429.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. 2019. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*.

Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. 2022. Lst: Ladder side-tuning for parameter and memory efficient transfer learning. *ArXiv*, abs/2206.06522.

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*.

Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax.

Yaqing Wang, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. 2022. Adamix: Mixture-of-adapter for parameter-efficient tuning of large language models. *arXiv preprint arXiv:2205.12410*, 1(2):4.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Kaiyan Zhang, Ning Ding, Biqing Qi, Xuekai Zhu, Xinwei Long, and Bowen Zhou. 2023a. Crash: Clustering, removing, and sharing enhance fine-tuning without full large language model. In *Empirical Methods in Natural Language Processing*.

Minjia Zhang and Yuxiong He. 2020. Accelerating training of transformer-based language models with progressive layer dropping. *Advances in Neural Information Processing Systems*, 33:14011–14023.

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 *International Conference on Learning Representations*. Openreview.

Ruiyi Zhang, Rushi Qiang, Sai Ashish Somayajula, and Pengtao Xie. 2024. Autolora: Automatically tuning matrix ranks in low-rank adaptation based on meta learning. *arXiv preprint arXiv:2403.09113*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2023c. Opt: Open pre-trained transformer language models. *URL https://arxiv.org/abs/2205.01068*, 3:19–0.

Ligeng Zhu, Lanxiang Hu, Ji Lin, and Song Han. 2024. Lift: Efficient layer-wise fine-tuning for large model models.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2024. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems*, 36.

Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

A Appendix

A.1 Code and Reproducibility

Algorithm 1: IST, PyTorch-like

```
import peft, transformers
from ist import IST
# initialize the pre-trained model and PEFT
  modules
peft_model = get_peft_model()
# initialize IST as callback function
ist_callback = IST()
# adopt IST in Trainer with one modification
trainer = transformers.Trainer(model=peft_model,
  callbacks=[ist_callback]))
trainer.fit()
```

Our code is based on the LLM-Adapter library* (Hu et al., 2023), a benchmark library for parameter-efficient fine-tuning (PEFT). To facilitate reproducibility, we have included the code, along with training scripts and instructions, in the supplementary material. Notably, our IST method is orthogonal to most PEFT methods and can be readily incorporated into the training process. As demonstrated in algorithm 1, our method requires only a single line of modification to the trainer based on the Hugging Face Transformers library† and Peft library‡. Please refer to the code for more details.

A.2 PEFT Overview

Method	Prompt	Repara	Series	Parallel
Prompt Tuning (Lester et al., 2021)	√			
Prefix-Tuning (Li and Liang, 2021)	√			
LoRA (Hu et al., 2021)		√		
KronA (Edalati et al., 2022)		√		
DoRA (Liu et al., 2024)		√		
Adapters (Houlsby et al., 2019)			√	
AdaMix (Wang et al., 2022)			√	
SparseAdapter (He et al., 2022b)			√	
LeTS (Fu et al., 2021)			√	
Parallel Adapter (He et al., 2022a)				√
LST (Sung et al., 2022)				√
MAM Adapter (He et al., 2021)	√	√	√	
UniPELT (Mao et al., 2021)	√	√	√	
Compacter (Henderson et al., 2021)		√	√	
S4-model (Chen et al., 2023)	√	√		

Table 6: The PEFT methods are categorized based on the four common basic methods. "Prompt" represents prompt-based learning methods, "Repara" denotes reparametrization-based methods, "Series" is Series Adapters, and "Parallel" represents Parallel Adapters.

According to Hu et al. (2023) and Han et al. (2024), existing parameter-efficient fine-tuning (PEFT) methods can be roughly categorised into four types as shown in Table 6. In the following, we provide a brief overview of three layer-based PEFT methods used in our study: reparametrization-based methods, series adapters, and parallel adapters.

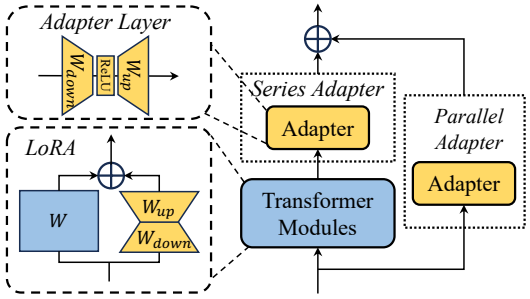


Figure 5: Most existing PEFT approaches employ a layer-based design, consistently adding learnable modules or parameters to each layer of the transformer modules, including the Multi-Head Self-Attention (MHSA) and Feed-Forward Network (FFN).

Parallel Adapters. Parallel adapters focus on incorporating additional learnable modules in parallel with distinct sublayers within the backbone model. The Parallel Adapter can be formulated as follows:

$$H_o \rightarrow H_o + f(H_i W_{down}) W_{up}. \quad (11)$$

Reparametrization-based method. This type of method aims to transform network weights using a low-rank technique. We take LoRA (Hu et al., 2021) as an example of Reparametrization-based learning, which can be formulated below:

$$H_o = H_i W_0 + H_i \Delta W = H_i W_0 + H_i BA, \quad (12)$$

where H_i and H_o are the input and output of a sublayer module (e.g., Linear), $W_0 \in \mathbb{R}^{d \times d}$ can be any linear weight in the pre-trained LLM, $B \in \mathbb{R}^{r \times d}$ and $B \in \mathbb{R}^{d \times r}$ are lower-rank learnable matrix to approximate ΔW. $r \ll d$ is the pre-defined rank for LoRA.

Series Adapters. Series adapters involve incorporating additional learnable modules in a sequential manner within a specific sublayer. Series Adapter can be formulated as follows:

$$H_o \rightarrow H_o + f(H_o W_{down}) W_{up}, \quad (13)$$

where H_o is the output of a specific layer like MLP layer, $f(\cdot)$ is a non-linear function like ReLU, $W_{down} \in \mathbb{R}^{d \times r}$ and $W_{up} \in \mathbb{R}^{r \times d}$ form a bottleneck MLP to save learnable parameters.

*https://github.com/AGI-Edgerunners/LLM-Adapters
†https://github.com/huggingface/transformers
‡https://github.com/huggingface/peft

Dataset	# Train	# Test	Answer
Commonsense	170K	-	-
BoolQ	9.4K	3,270	Yes/No
PIQA	16.1K	1,830	Option
SIQA	33.4K	1,954	Option
HellaSwag	39.9K	10,042	Option
WinoGrande	63.2K	1,267	Option
ARC-e	1.1K	2,376	Option
ARC-c	2.3K	1,172	Option
OBQA	5.0K	500	Option
Math10K	10K	-	-
GSM8K	8.8K	1,319	Number
AQuA	100K	254	Option
MAWPS	-	238	Number
SVAMP	-	1,000	Number

Table 7: The statistics of datasets for evaluation. # Train and # Test denote the number of training and test samples respectively.

As shown in the Figure 5, the three PEFT methods mentioned above all utilize the layer-based design, i.e., adding identical learnable modules or parameters to each layer of the pre-trained LLM.

It is important to note that we did not include the prompt-based method in our comparison because original prompt tuning (Lester et al., 2021) is not a layer-based method; rather, it adds learnable soft prompts at the input layer. Furthermore, while some advancements in prompt tuning are layer-based, such as Prefix Tuning (Li and Liang, 2021), which independently adds soft prompts to the hidden states at all layers, they do not align with our design. This misalignment occurs because our proposed response suppression operates on the output of a PEFT method conditioned on the input, whereas prompt-based methods produce an output that is not conditioned on the input.

A.3 Experimental Details

A.3.1 Dataset Statistics

Detailed dataset statistics can be referred to Table 7. Note that we trained on Commonsense and Math10K for commonsense reasoning and arithmetic reasoning, respectively. During testing, we evaluated the predefined test sets of each dataset.

A.3.2 Hyperparameters

Detailed hyperparameter settings are provided in Table 8 and Table 9. For PEFT training, we adhere to the settings outlined by LLM-Adapter Library (Hu et al., 2023), with the exception of the learning rate. For IST, we consistently set N_u to 25% of the layers for the fine-tuning loop, N_v to 50% of the layers for the importance updating loop, and β to 0.25. Additionally, μ is set to 10 and 100 for the Commonsense Reasoning task and the Arithmetic Reasoning task, respectively.

A.4 Additional Experiments

A.4.1 Importance Updating Rate μ

The updating rate of importance is associated with several hyperparameters, such as T_c, N_c, N_v, and μ. To narrow the hyperparameter search space and reduce the complexity of using IST, we fixed most hyperparameters, setting T_c to 10, N_c to 3, and N_v to half the number of layers. We then adjusted the importance updating rate μ to match with the dynamics of PEFT fine-tuning. This parameter is largely dependent on the maximum number of training iterations. If μ is too small, the method will approximate a random strategy. Conversely, if μ is too large, the method will tend to train only a fixed set of layers. As shown in Table 10, we consider μ values of [0.1, 1, 10, 100, 1000]. μ is a parameter that exhibits insensitivity, indicating the robustness of our method.

A.4.2 Response Suppression Factor β

Table 11 illustrates the effect of varying the response suppression factor β on training a LLaMA 7B model using the commonsense dataset. We evaluated among four values: [0, 0.1, 0.25, 0.5]. When the factor is set to 0, which is equivalent to dropping the PEFT modules within the layer, it does not adequately reflect the importance of PEFT. Increasing the factor enhances performance, peaking at 0.25. This indicates that compared to removing the PEFT modules, suppressing its output better captures its influence on the loss. However, further increasing the factor to 0.5 results in diminished effectiveness, likely due to reduced variation in loss. These findings suggest that a relatively small, non-zero factor is optimal for accurately estimating the PEFT module's impact on loss.

A.4.3 Adaptability to SoTA PEFT method

To demonstrate the versatility of the IST method, we integrated IST into a recent LoRA variant called DORA (Liu et al., 2024), which decouples the low-rank component into direction and magnitude, yielding better performance. As shown in Table 12, our method can enhance the DoRA method in

Hyperparameters (LoRA + IST)	Commonsense Reasoning					Arithmetic Reasoning
	LLaMA_{7B}	LLaMA_{13B}	GPT-J_{6B}	BLOOMz_{7B}	LLAMA3_{8B}	LLAMA3_{8B}
Rank r	32					32
α	64					64
Dropout	0.05					0.05
Optimizer	AdamW					AdamW
LR	2e-4					1e-4
LR Scheduler	Warmup Steps					Warmup Steps
Batch size	16					16
Warmup Steps	100					100
Epochs	3					3
Where	{Q, K, V, Up, Down}					{Q, K, V, Up, Down}
μ	10					100
β	0.25					0.25
N_L, N_u, N_v	{32, 8, 16}	{40, 10, 20}	{28, 7, 14}	{30, 8, 15}	{32, 8, 16}	{32, 8, 16}

Table 8: Hyperparameter configurations of IST for LLaMA-7B/13B, GPT-J 6B, BLOOMz 7B, and LLaMA3-8B with LoRA.

Hyperparameters	Series Adapter + IST		Parallel Adapter + IST	
	LLaMA_{7B}	LLaMA_{13B}	LLaMA_{7B}	LLaMA_{13B}
Bottleneck Size	256			
Optimizer	AdamW			
LR	2e-4			
LR Scheduler	Warmup Steps			
Batch size	16			
Warmup Steps	100			
Epochs	3			
Where	{Up, Down}		{Up, Gate}	
μ	10			
β	0.25			
N_L, N_u, N_v	{32, 8, 16}	{40, 10, 20}	{32, 8, 16}	{40, 10, 20}

Table 9: Hyperparameter configurations of IST for LLaMA-7B/13B on commonsense reasoning tasks with series and parallel adapters.

Random	IST				
	μ=0.1	μ=1	μ=10	μ=100	μ=1000
75.8	75.7	75.9	**76.5**	74.8	73.9

Table 10: Sensitivity of importance updating rate μ.

Commonsense Reasoning tasks without any loss of performance, while also requiring less memory and computational resources. This efficiency is achieved by explicitly training only a subset of all layers, highlighting the general applicability of our proposed IST.

A.5 Time Consumption

To accurately estimate the training time, we randomly sampled prompts from the Alpaca dataset and limited the maximum output token length to 1024. We used LoRA on LLaMA-7B with a rank of 32 as our baseline. Additionally, we employed the LISA (Pan et al., 2024) method, which randomly selects two transformer layers for updating. We conducted 140 iterations and averaged the forward and backward times of the middle 100 iterations to obtain a stable time estimate during training. As shown in Table 13, LISA reduces the forward time compared to LoRA due to the absence of additional parameters for inference, while it increases the backward time. Conversely, IST maintains the forward pass time but reduces the backward time by approximately 10%. Despite this, IST requires an additional three forward passes every 10 fine-tuning loops for importance updating. Consequently, after 100 iterations, the total time consumption for IST becomes comparable to that of LISA and slightly higher than LoRA.

Method	Results
Baseline	74.7
IST with $\beta = 0$	75.0
IST with $\beta = 0.1$	76.3
IST with $\beta = 0.25$	**76.5**
IST with $\beta = 0.5$	75.3

Table 11: Effect of response suppression factor β within IST for LLaMA-7B on commonsense reasoning tasks with LoRA.

Model	PEFT	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
ChatGPT	-	73.1	85.4	68.5	78.5	66.1	89.8	79.9	74.8	77.0
$\text{LLaMA3}_{\text{8B}}$	$\text{DoRA}_{ICML2024}$	74.6	89.3	79.9	95.5	85.6	90.5	80.4	85.8	**85.2**
	DoRA + IST	74.0	89.2	80.2	95.0	86.2	90.3	81.2	85.6	**85.2**

Table 12: Adaptability to the latest LoRA-variant method called DoRA (Liu et al., 2024). Our approach can reduce memory consumption without compromising accuracy.

	LoRA	LISA	LoRA + IST
Forward time per iter. (ms)	135	**101**	135
Backward time per iter. (ms)	184	225	**150**
Time consumption per 100 iter. (s)	**31.9**	32.6	32.6

Table 13: Comparison of Training Times. All results were obtained using one Nvidia GTX 4090 GPU.