Not All Adapters Matter: Selective Adapter Freezing for Memory-Efficient Fine-Tuning of Language Models

Hyegang Son[*1], Yonglak Son[*1], Changhoon Kim[†‡2,3], and Young Geun Kim[‡1]
[1]Korea University, {hyegang_son, yonglak_son, younggeun_kim}@korea.ac.kr
[2]Arizona State University, kch@asu.edu
[3]Soongsil University, changhooon.kim@gmail.com

Abstract

Transformer-based large-scale pre-trained models achieve great success. Fine-tuning is the standard practice for leveraging these models in downstream tasks. Among the fine-tuning methods, adapter-tuning provides a parameter-efficient fine-tuning by introducing lightweight trainable modules while keeping most pre-trained parameters frozen. However, existing adapter-tuning methods still impose substantial resource usage. Through our investigation, we show that each adapter unequally contributes to both task performance and resource usage. Motivated by this insight, we propose Selective Adapter FrEezing (SAFE), which gradually freezes less important adapters early to reduce unnecessary resource usage while maintaining performance. In our experiments, SAFE reduces memory usage, computation amount, and training time by 42.85%, 34.59%, and 11.82%, respectively, while achieving comparable or better task performance compared to the baseline. We also demonstrate that SAFE induces regularization effect, thereby smoothing the loss landscape, which enables the model to generalize better by avoiding sharp minima.

1 Introduction

Large-scale pre-trained language models (PLMs) have manifested superior performance in various tasks (Kenton and Toutanova, 2019; Liu et al., 2019; Radford et al., 2019; Yang et al., 2019). However, training PLMs from the scratch is time-consuming and resource-intensive. Common practice has been hence to fine-tune the large-scale pre-trained models by adapting all the parameters with the downstream tasks, i.e., full parameter fine-tuning (full-tuning).

Recently, Parameter-Efficient Fine-Tuning (PEFT), which focuses on optimizing a small fraction of parameters for downstream tasks, is receiving much attention (Houlsby et al., 2019; Lester et al., 2021; Li and Liang, 2021; Liu et al., 2022, 2023). Among various PEFT strategies, adapter-tuning has emerged as a prevalent method. It integrates lightweight modules, termed adapters, into each layer of PLMs and only tunes the adapters with the downstream tasks. As shown in Figure 1(a), the adapter-tuning methods (Houlsby et al., 2019; Pfeiffer et al., 2021; Zaken et al., 2022; Hu et al., 2021; Zhang et al., 2022), significantly reduce the number of trainable parameters compared to the full-tuning while exhibiting better performance on a downstream task.

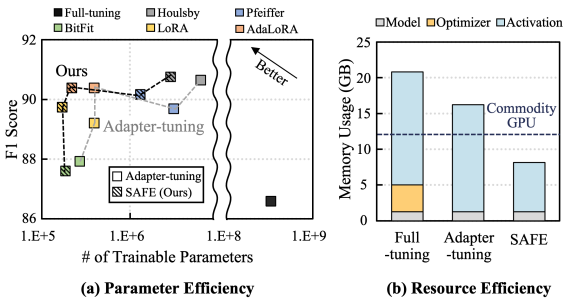


Figure 1: Comparison between full-parameter fine-tuning, adapter-tuning and our proposed SAFE on the $\text{BERT}_{\text{large}}$ model with SQuAD dataset. SAFE significantly reduces memory usage while providing comparable accuracy to adapter-tuning.

As adapter-tuning reduces the number of trainable parameters, it is also expected to reduce the resource (i.e., memory) usage accordingly. Unfortunately, parameter-efficiency does not always translate into resource-efficiency. As shown in Figure 1(b), although adapter-tuning significantly reduces the number of trainable parameters (by 99.37%, on average) compared to the full-tuning, the memory usage is not much reduced (only by 22.19%, on average). This is because adapter-tuning does not reduce activation memory (i.e., intermediate values for reuse during backpropagation) which account for 76.00% of memory usage

[*]Equal contribution.
[†]Work completed as part of Ph.D. research at ASU.
[‡]Corresponding authors.

Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 9479–9496
April 29 - May 4, 2025

— it only reduces optimizer memory (e.g., gradients and momentum vectors). Considering the remarkable increase in model size compared to the modest increase in GPU memory capacity, adapter-tuning methods still face challenges in terms of memory efficiency. For example, fine-tuning of a LLaMA-65B (Touvron et al., 2023) requires more than 780GB of GPU memory. As shown in Figure 1(b), enabling resource-efficient fine-tuning may enhance accessibility of fine-tuning to researchers and end-users, by reducing memory requirements below the capacity of commodity GPU memory.

According to previous work, the activation memory mostly depends on the backpropagation length (Chen et al., 2016; Rhu et al., 2016), which is determined by the number of adapters trained during the backward pass. Hence, to reduce the activation memory, it is crucial to reduce the number of training adapters. However, merely reducing the number of training adapters degrades accuracy. Here, a pivotal research problem arises:

Can we reduce the number of training adapters without sacrificing accuracy?

To answer the question, we analyze the impact of individual adapters on the accuracy and resource usage of training (Figure 2 in Section 3). We observe that some adapters are being trained, even after they finish contributing to the accuracy improvement, occupying memory. Thus, it is possible to stop training (i.e., freezing) such adapters early if they do not contribute to the adaptation for a downstream task, de-allocating their activation memory. We also observe that such early freezing can even lead to the regularization effect on the model (Fu et al., 2023), improving the accuracy.

In this paper, we propose **SAFE** (**S**elective **A**dapter **FrE**ezing), which adaptively freezes adapters in the early epochs of training. In each epoch, SAFE identifies adapters that contribute relatively less to the accuracy improvement by using an importance score (Kornblith et al., 2019). It then freezes the adapters whose importance score is lower than a pre-defined threshold, reducing the memory usage and accelerating training time. By early freezing less important adapters, SAFE induces regularization effect on the model being trained, leading to a flatter loss surface. This is beneficial for finding an optimal point with higher generalization performance while optimizing neural network. In our evaluation, SAFE significantly reduces the average memory usage and TFLOPs by 46.89% and 51.73%, respectively, across various models and downstream tasks without compromising accuracy compared to the baseline, LoRA (Hu et al., 2021). SAFE even improves the accuracy for some tasks, compared to LoRA, by up to 4.33% while reducing memory usage by 53.60%, by inducing the regularization effect.

In summary, our key contributions include:

- We uncover that adapters exhibit varying degrees of contribution to model adaptation and resource usage (Section 3).
- Motivated by this observation, we propose SAFE, a novel approach that enables resource-efficient fine-tuning by selectively early-freezing less important adapters (Section 4).
- Our evaluation on various downstream tasks demonstrates that SAFE not only achieves comparable or even better task performance to baselines but also significantly reduces resource usage by inducing the regularization effect on the model (Section 5).

2 Related Work

Parameter-Efficient Fine-Tuning: To efficiently adapt large-scale PLMs to downstream tasks, many adapter-tuning methods (Chen et al., 2023; He et al., 2023; Hu et al., 2021; Houlsby et al., 2019; Karimi Mahabadi et al., 2021; Liu et al., 2022) have been proposed. In general, adapter-tuning methods inject small, trainable, and task-specific adapter modules into each transformer layer of a pre-trained model. Given a pre-trained weight matrix $W_0 \in \mathbb{R}^{d\times k}$ and input $x \in \mathbb{R}^{k\times 1}$, the weight update of adapter-tuning is expressed as $W_0 + \Delta W$. During training, W_0 is frozen and does not receive gradient updates, while ΔW contains trainable parameters. For $h = W_0 x$, The modified forward pass in adapter-tuning yields:

$$h = W_0 x + \Delta W x. \quad (1)$$

To further improve parameter efficiency of adapter-tuning, AdaLoRA (Zhang et al., 2022) adaptively adjusts the number of trainable parameters among adapters according to their importance score — it reduces the number of trainable parameters for less important adapters. However, the adapter-tuning methods still use a large amount of memory, as shown in Figure 1, since they do not reduce the activation memory which accounts for a large portion of memory usage.

Pruning LLM Model Parameters: To reduce the model memory of fine-tuning, two categories of

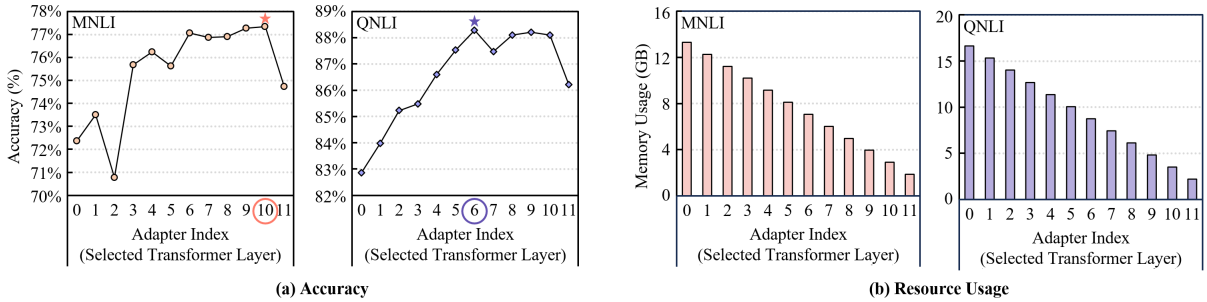


Figure 2: (a) Accuracy and (b) Resource usage of adapter-tuning by injecting an adapter into each transformer layer of $\text{BERT}_{\text{base}}$ model on MNLI and QNLI datasets from GLUE.

pruning methods have been proposed (Liang et al., 2021): structured pruning and unstructured pruning. Structured pruning methods remove grouped parameters (e.g., channels, layers) from the LLM. However, they usually degrade the accuracy. Furthermore, they have a limitation in terms of the compression ratio because of the low flexibility. LLM-Pruner (Ma et al., 2023) compensates the accuracy drop coming from pruning, by employing post-training.

To overcome the limitation of structured pruning, unstructured pruning methods remove partial values of weight matrices regardless of their structures (Li et al., 2022b; Frantar and Alistarh, 2023). However, unstructured pruning also degrades the accuracy.

Resource Efficient Fine-Tuning: Several works have tried to target resource efficient fine-tuning. AdapterDrop (Rücklé et al., 2021), randomly excludes partial adapters from each training step. However, it cannot de-allocate the activation memory for the adapters, because of the random selections — an adapter excluded from training in a step can be included in training in the following steps. SparseAdapter(He et al., 2022) applies unstructured pruning to the adapters. However, it also does not reduce the actual memory usage — this is because the weight matrices pruned with zero values still need to be fully allocated in the memory. LoRAPrune(Zhang et al., 2023) employs structured pruning for LoRA. Unfortunately, the aforementioned methods usually have an adverse impact on the accuracy. MEFT (Liao et al., 2024) applies a reversible model to PEFT. By using the reversible network, MEFT calculates activations with accumulated outputs of layers, without saving the intermediate activations reducing the activation memory. However, calculations of the activations severely degrades the training time performance.

Different from the previous works, this work freezes less important adapters in early steps of training. Since the frozen adapters can only be used for the forward pass, early freezing of less important adapters can effectively reduce the back-propagation length as well as the activation memory. Moreover, it induces regularization effect on the model, improving its accuracy.

3 Motivation

In this section, we present a pivotal research question for resource-efficient fine-tuning.

> **RQ**: Do all adapters contribute equally to the process of adaptation?

To answer this question, we analyze the impact of adapters injected into each transformer layer on accuracy and resource efficiency. We measure the accuracy and memory usage of $\text{BERT}_{\text{base}}$ model on MNLI and QNLI dataset from GLUE (Wang et al., 2018), by attaching an adapter to each transformer layer one-by-one. Figure 2(a) and (b) show the measured accuracy and memory usage respectively — the x-axis indicates the index of transformer layer that the adapter is injected into.

As shown in Figure 2(a), each adapter has different impact on the accuracy, and the importance of each adapter varies depending on the downstream task. In addition, despite uniform counts of trainable parameters, resource usage decreases for adapters closer to the output layer, as depicted in Figure 2(b). These observations point to the possibility that adapters in early layers contribute less to task adaptation, even though they require considerable resources. In other words, if we selectively deactivate less impactful adapters, it is possible to co-optimize the resource efficiency and accuracy.

To further analyze changes of the feature representations for each adapter throughout the training process, we quantify the representation similarity

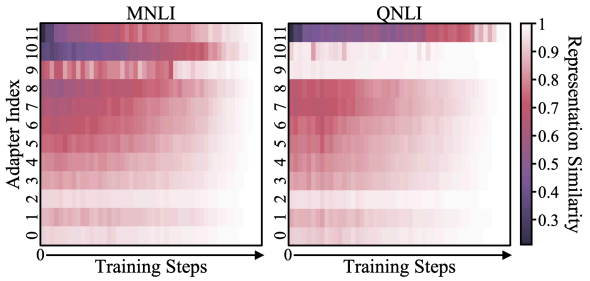


Figure 3: Visualization of representation similarity between the trained model and the model at different training steps during adapter-tuning of the $\text{BERT}_{\text{base}}$ model on the MNLI and QNLI datasets from GLUE.

between adapters in each training step and those in the final model (which we obtained after the convergence of fine-tuning). We quantify the representational similarity using Centered Kernel Alignment (CKA) by referring to previous works (Li et al., 2022a). Figure 3 visualizes the representational similarity measured throughout the training process for each adapter for $\text{BERT}_{\text{base}}$ model on MNLI and QNLI dataset from GLUE (Wang et al., 2018) — lighter the color becomes, higher the feature representation similarity is.

As shown in Figure 3, even in the early training steps, the feature representations of several adapters are almost the same as those of the final model — similar patterns are observed in other models and datasets. This means that those adapters are already representing the features that should be represented by the final model, and thus they may not further be adapted for the downstream task in the rest of the training steps. This is why such adapters are less contributing to the accuracy improvement of the model on the downstream tasks in Figure 2. One intuition is that lower adapters generally learn basic understanding of the input data, such as data bias and structural characteristics of the data, while adapters closer to the output build features unique to different tasks (Houlsby et al., 2019). Motivated by the observation where not all adapters consistently contribute to adaptation, in the next section, we propose a selective adapter freezing method which preemptively freezes adapters that are relatively less important for each task.

4 Selective Adapter Freezing (SAFE)

In this section, we propose a selective adapter freezing method, SAFE. SAFE adaptively freezes less important adapters in the early training steps, in order to reduce unnecessary computation and memory usage without compromising the accuracy.

Figure 4 shows the overview of SAFE. SAFE consists of two stages: warm-up stage and freezing stage. In the warm-up stage, SAFE performs several epochs of fine-tuning while monitoring the feature representation changes (i.e., importance score) of the adapters (Section 4.1). If the important score of all adapters is not much changed for consecutive epochs, SAFE enters the freezing stage. In the freezing stage, SAFE gradually freezes adapters that contribute less to the adaptation, based on the importance score (Section 4.2). By early freezing less important adapters, SAFE induces regularization effect on model (Section 4.3), leading to better performance.

4.1 Importance Score

In the warm-up stage[1], we identify less important adapters by monitoring the feature representation changes of the adapters. To capture the feature representation changes of the adapters, SAFE uses Centered Kernel Alignment (CKA), which is a representative metric for representation similarity — similar practice has been used in previous works (Neyshabur et al., 2020; Raghu et al., 2021). It calculates CKA between the activation of a layer adapted with an adapter and that of the original layer as:

$$\text{CKA}_i(X_i, Y_i) = \frac{\|Y_i^T X_i\|_F^2}{\|X_i^T X_i\|_F \|Y_i^T Y_i\|_F}, \quad (2)$$

where X_i and Y_i are the activations of a layer that is adapted with an adapter and the original layer, respectively, i = Index of Layer, and $\|\cdot\|_F^2$ represents the square of the Frobenius norm of a matrix.

Higher CKA value indicates that the feature representation of a layer is still similar with that of the original one. To this end, SAFE calculates the importance score of an adapter as:

$$Imp(Adapter_i) = 1 - \text{CKA}_i(X_i, Y_i) \quad (3)$$

4.2 Adapter Freezing

In the freezing stage, SAFE gradually freezes adapters based on their importance score. At t_w-th epoch, SAFE compares the importance score of adapters with threshold τ_T. If the importance score of an adapter is lower than τ_T, SAFE identifies the adapter as a freezing candidate. After identifying

[1]We define the number of warm-up epochs as the epoch at which the importance score of all adapters change by less than 5% for consecutive epochs.

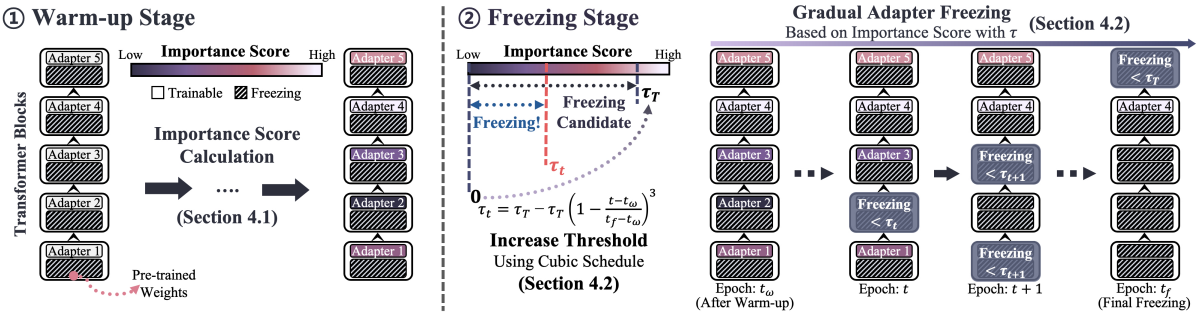


Figure 4: Design overview of Selective Adapter Freezing (SAFE). At the warm-up stage, SAFE identifies important adapters by calculating importance score. At the freezing stage, SAFE gradually freezes the adapter based on their importance score with moving threshold τ by following a cubic schedule.

freezing candidates, SAFE freezes them based on a moving threshold until t_f-th epoch — it increases the threshold from 0 to τ_T[2] between t_w-th and t_f-th epochs following a cubic schedule as (Zhang et al., 2022):

$$\tau_t = \begin{cases} 0 & 0 \le t < t_w, \\ \tau_T - \tau_T \left(1 - \frac{t-t_w}{t_f-t_w}\right)^3 & t_w \le t < t_f, \\ \tau_T & \text{o.w.} \end{cases} \quad (4)$$

where t is the current epoch, t_w is the number of initial warm-up epochs, t_f is the number of final freezing epochs. The cubic schedule follows a non-linear growth pattern, enabling rapid exploration of the search space during initial stages and gradually slowing down to reliably converge to the target point. We leverage these advantages by freezing more less-important adapters in the early stages. As the number of trainable parameters in the model decreases, SAFE gradually reduces the number of freezing adapters, reaching the threshold τ_T stably.

4.3 Regularization Effect of SAFE

By selectively freezing less critical adapters, SAFE induces a regularization effect within the model. In transformer-based PLM $\mathcal{N}_0$, each of the l transformer blocks T_l is equipped with a distinct set of parameters θ_l^0 for $l \in \{1, \ldots, n\}$. To reduce the computational overhead of fine-tuning all parameters θ_l^0, lightweight adapters $\Delta\theta_l$ are introduced. To clarify how introducing adapters contribute to performance enhancements, Fu et al. (2023) formalize the optimization function as follows:

$$\min_{\theta} \mathcal{L}(\theta) + \|(I - M)(\theta - \theta^0)\|^2, \quad (5)$$

where $\theta = \theta^0 + M\Delta\theta$ and $M \in \{0, 1\}^{m \times m}$, with $m = \dim(\theta)$, serves as a diagonal matrix for selective parameter adjustment. Each diagonal element $M_{ii} \in \{0, 1\}$ indicates whether the corresponding parameter of $\Delta\theta_i$ is active (1) or inactive (0), with all off-diagonal elements M_{ij} set to 0. The regularization term is crucial for explaining how parameter constraints introduced by adapters can enhance model performance on downstream tasks. The $rank(M)$ is bounded by m, reflecting full capacity for parameter adaptation within each transformer block. However, such an approach can lead to excessive computation (See Figure 1). In contrast, our study explores the implications of constraining the $rank(M)$ to a reduced upper bound from m to the number of trainable parameters following the proposed freezing algorithm in Section 4.2. For instance, in our motivational analysis in Section 3, where we limit the number of trainable parameters to one adapter per layer, the $rank(M)$ is bounded by $\frac{m}{l}$ by selectively activating ΔW_l for T_l. This constraint not only optimizes computational efficiency but also preserves the adaptability essential for superior performance on downstream tasks, as evidenced by our empirical results detailed in Section 5.3.

5 Experiments

5.1 Experimental Setting

Models: We assess the fine-tuning efficacy of SAFE using state-of-the-art transformer-based models, including $\text{BERT}_{\text{base}}$, $\text{BERT}_{\text{large}}$ (Kenton and Toutanova, 2019), $\text{RoBERTa}_{\text{base}}$, $\text{RoBERTa}_{\text{large}}$ (Liu et al., 2019), $\text{GPT-2}_{\text{medium}}$, $\text{GPT-2}_{\text{large}}$ (Radford et al., 2019), and $\text{LLaMA-2}_{\text{7B}}$ (Touvron et al., 2023).

Datasets: The aforementioned models are evalu-

[2]We empirically determine τ_T and final freezing epochs t_f based on extensive experiments with various models and datasets.

Table 1: Experimental results with $\text{BERT}_{\text{large}}$ on natural language understanding tasks from the GLUE benchmark. SAFE significantly reduces memory usage while achieving GLUE score comparable to the baseline. Note that we report memory usage and computation costs on RTE task.

	CoLA	SST-2	MNLI	RTE	QQP	MRPC	QNLI	STS-B	Avg.	Memory	Computation
	Matthews corr	Accuracy	Accuracy	Accuracy	Accuracy	F1 Score	Accuracy	Pearson corr		(GB)	(TFLOPs)
LoRA	65.24	93.65	85.40	72.66	90.49	87.90	90.06	91.88	84.66	20.35	46,698
+ Zhang et al.	61.78	91.97	84.61	73.18	89.20	86.83	90.00	87.50	83.13	11.20	35,415
+ AdapterDrop	64.24	92.54	85.19	73.38	89.02	86.51	91.51	91.39	84.22	20.35	35,114
+ SparseAdapter	65.25	92.66	85.19	74.10	90.43	88.50	91.61	91.99	84.97	20.35	46,698
+ LoRAPrune	63.03	91.54	83.97	70.59	88.28	87.74	84.13	86.48	81.97	10.37	25,137
+ MEFT	64.57	92.66	84.33	72.92	88.70	88.30	90.98	90.15	84.08	11.15	88,363
+ SAFE	65.26	92.78	85.41	74.10	89.96	88.84	91.78	91.80	**84.99**	**12.11**$_{40.47\%\downarrow}$	**30,285**$_{35.15\%\downarrow}$

ated across various tasks that span a broad spectrum of NLP applications, including Natural Language Understanding (NLU), Question Answering (QA), and Natural Language Generation (NLG). Initially, we utilize eight datasets from the General Language Understanding Evaluation (GLUE) (Wang et al., 2018) which comprises two single-sentence classification tasks, three similarity and paraphrase tasks, and four natural language inference tasks. Furthermore, we conduct experiments on the SQuAD dataset (Rajpurkar et al., 2016) with both BERT and RoBERTa model families. Decoder-only models such as $\text{GPT-2}_{\text{large}}$ are also tested to determine if SAFE maintains its effectiveness in the E2E NLG Challenge (Novikova et al., 2017). Finally, to investigate the scalability to larger models, we evaluate SAFE on the large language model ($\text{LLaMA-2}_{\text{7B}}$) and the WikiText-2 dataset (Merity et al., 2022). Detailed dataset descriptions are available in Appendix C.3.

Baselines: To evaluate the effectiveness of SAFE, we benchmark against state-of-the-art PEFT method, LoRA (Hu et al., 2021). We compare SAFE with five effective resource efficient fine-tuning methods, a previous work (Zhang et al.), AdapterDrop (Rücklé et al., 2021), SparseAdapter (He et al., 2022), LoRAPrune (Zhang et al., 2023), and MEFT (Liao et al., 2024). SAFE's performance is further compared with four other PEFT methods such as Houlsby (Houlsby et al., 2019), Pfeiffer (Pfeiffer et al., 2020), BitFit (Zaken et al., 2022), and the adaptive method AdaLoRA (Zhang et al., 2022) to demonstrate its versatility and applicability across different adapter-tuning frameworks (see Figure 1 and Appendix B for detailed results). Comprehensive details on the experimental setup and hyperparameters, such as training epochs and batch sizes, can be found in Appendix C.

5.2 Main results

5.2.1 Natural Language Understanding

Table 1 shows the results of different methods on GLUE tasks. Since SAFE selectively freezes 51.04% of less important adapters early throughout the training process, SAFE significantly reduces memory usage by 40.47%, from 20.35GB (LoRA) to 12.11GB, and decreases computation costs (FLOPs) by 35.15%. Even with such improvements in resource efficiency, SAFE improves the average GLUE score from 84.66 (LoRA) to 84.99 — this is because SAFE induces a regularization effect on less-important adapters improving generalization performance of the model (see Section 5.3).

Figure 5 shows the freezing patterns of $\text{BERT}_{\text{large}}$ fine-tuned with SAFE — we observe similar patterns for other tasks. We find that SAFE tends to freeze adapters more in layers closer to the input layer. Such behavior aligns with empirical observations presented in Figure 2 where adapters closer to the output layer need further adaptation to the downstream tasks compared to those in earlier layers, contributing more to model performance.

Compared to a previous work (Zhang et al.), which attaches adapters only to the upper layers, SAFE achieves 2.24% higher GLUE score. This is because some adapters attached to lower layers keep contributing to the adaptation even in later training epochs, as shown in Figure 5 (e.g., 2nd and 5th adapters in QNLI task). Compared to AdapterDrop, SAFE provides up to 2.69% higher score (MRPC) while reducing memory usage by 49.47% (0.91% higher GLUE score and 40.47% reduced memory usage on average). This is because AdapterDrop reduces computation costs (FLOPs) by ran-

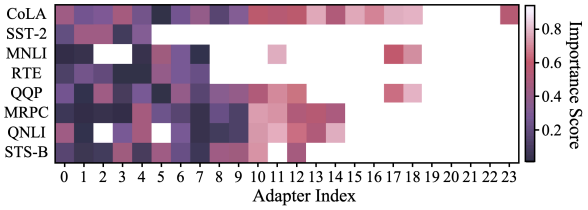


Figure 5: The freezing patterns when fine-tuning $\text{BERT}_{\text{large}}$ on GLUE with SAFE. Colors indicate adapters that are frozen, while white represents an adapter that is not frozen — the lighter the color is, the higher importance score is.

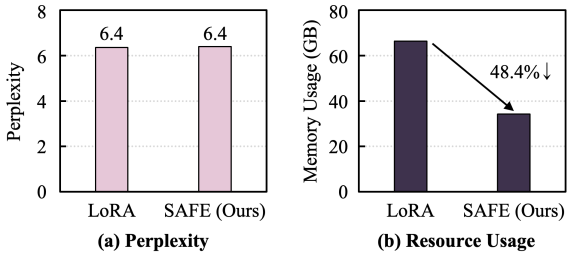


Figure 6: Comparison of (a) perplexity and (b) resource usage between LoRA and SAFE on the $\text{LLaMA-2}_{\text{7B}}$ model evaluated on the WikiText-2 dataset.

domly dropping adapters for each step, whereas SAFE selectively freezes less-important adapters preserving critical adapters and thus achieving better performance. AdapterDrop also does not lower memory usage because the memory allocated to the dropped adapters cannot be de-allocated for the next step — adapters dropped in a step may not be dropped in the next step. Compared to SparseAdapter, SAFE reduces memory usage by 40.47% and computation cost by 35.15% while providing comparable GLUE score. This is because SparseAdapter performs pruning of redundant parameters but uses unstructured pruning with masking, which does not actually improve resource efficiency. On the other hand, although LoRAPrune is effective to reduce the memory usage, it severely degrades the accuracy as it fully eliminates less important adapter weights through structural pruning. As a result, SAFE achieves 3.68% higher GLUE score compared to LoRAPrune. MEFT reduces the memory usage at the cost of more than 2x of the FLOPs and training time. This is because MEFT applies a reversible model to LoRA, which reduces memory usage by not caching intermediate activations, but causes substantial computation overhead due to recomputation. Overall, SAFE strikes a better accuracy/memory efficiency/training-time performance trade-off compared to SOTA methods.

Table 2: Experimental results on question answering task from the SQuAD dataset.

		F1 Score	Memory Usage (GB)	Computation (TFLOPs)
$\text{BERT}_{\text{base}}$	LoRA	86.99	5.95	611,295
	+ SAFE	**87.22**	**4.61**22.52%↓	**455,274**25.52%↓
$\text{BERT}_{\text{large}}$	LoRA	89.22	15.79	2,117,791
	+ SAFE	**89.72**	**7.44**52.88%↓	**869,624**58.94%↓
$\text{RoBERTa}_{\text{base}}$	LoRA	90.95	11.51	1,225,147
	+ SAFE	**91.16**	**7.80**32.23%↓	**808,239**34.03%↓
$\text{RoBERTa}_{\text{large}}$	LoRA	93.39	17.73	2,117,791
	+ SAFE	**94.13**	**3.56**79.92%↓	**245,541**88.41%↓

Table 3: Experimental results on natural language generation from the E2E NLG Challenge. For all metrics, higher is better. Note that we report memory usage reduction in blue.

		BLEU	NIST	METEOR	ROUGE-L	CIDEr	
$\text{GPT-2}_{\text{medium}}$	LoRA	68.91	8.68	46.48	71.33	2.47	
	+ SAFE	68.67	8.66	46.40	70.88	2.43	34.34%↓
$\text{GPT-2}_{\text{large}}$	LoRA	70.27	8.85	46.40	71.63	2.52	
	+ SAFE	70.26	8.87	46.58	71.68	2.53	25.50%↓

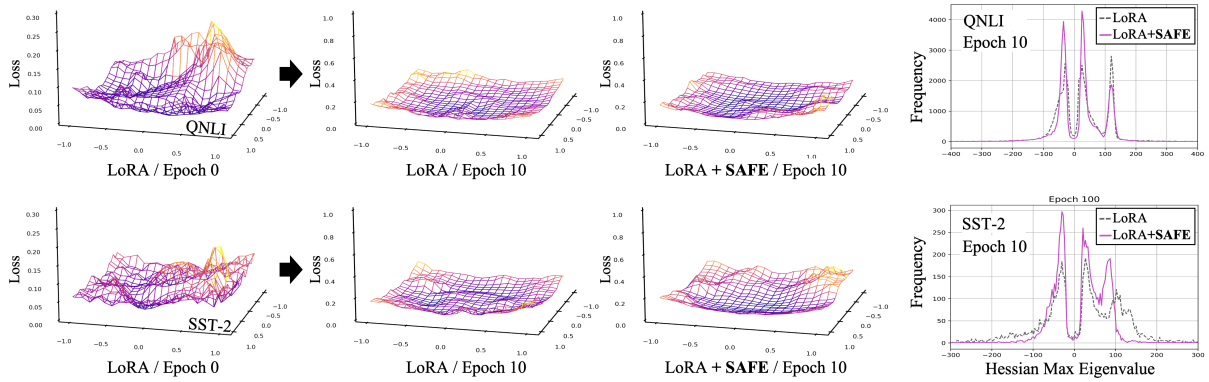
5.2.2 Question Answering

Table 2 shows the results of SQuAD dataset. SAFE consistently outperforms baseline under all settings. Notably, SAFE reduces memory usage and computation costs by up to 79.92% and 88.41% on $\text{RoBERTa}_{\text{large}}$ by freezing 91.67% of the adapters, while improving the F1 score from 93.39 (LoRA) to 94.13. This result also demonstrates that the benefits and effectiveness of SAFE are not restricted to specific model sizes, making it a valuable strategy for enhancing adapter-tuning outcomes across models of varying scales.

5.2.3 Natural Language Generation

Table 3 shows that SAFE prevails on natural language generation task with GPT-2. SAFE achieves comparable performance to LoRA across all metrics while significantly reducing memory usage. This result also demonstrates that SAFE is effective not only for encoder models but also works well with decoder models.

Figure 6 compares (a) perplexity and (b) memory usage between LoRA and SAFE on the $\text{LLaMA-2}_{\text{7B}}$ using the WikiText-2 dataset, to examine the applicability of SAFE on large language models. As shown in Figure 6, SAFE reduces memory usage by 48.37%, from 66.35GB (LoRA) to 34.20GB, without any degradation in model qual-



(a) Loss Landscape Visualization **(b) Hessian Eigenvalue Spectrum**

Figure 7: (a) Loss landscape demonstrates that SAFE yields a flatter loss surface compared to the baseline, as shown in the second and third columns. (b) Hessian eigenvalue spectrum analysis shows that the magnitude of the Hessian eigenvalues for SAFE is smaller than those for the baseline, indicating a flatter local curvature and potentially better generalization properties.

ity. This result implies that SAFE can significantly improve the scalability of large language models by reducing the resource usage, making it feasible to train these models on resource-constrained devices or with larger batch sizes.

5.3 Regularization Effect

To elucidate the underlying mechanisms behind SAFE's enhancements in model performance and memory efficiency, we conduct a detailed empirical analysis. We visualize and compare the loss landscapes of the baseline and SAFE. Additionally, we quantitatively evaluate the flatness of the loss surfaces by analyzing the spectrum of Hessian eigenvalues. This methodical approach allows us to substantiate the improvements attributed to SAFE, providing insights into its effectiveness in optimizing both performance and resource utilization.

Loss Landscape Analysis. The flatness of a loss landscape is a recognized indicator of the generalization ability of models (Jiang et al., 2020). Specifically, flatter landscapes are indicative of enhanced robustness to parameter perturbations (Xie et al., 2021), reduced model complexity (Blier and Ollivier, 2018), and improved generalization capabilities (Cha et al., 2021, 2022; Choromanska et al., 2015; Park and Kim, 2021; Wu and Su, 2023). To examine these properties, we employ a comparative visualization of the loss landscapes for LoRA and SAFE using the $\text{BERT}_{\text{base}}$ model on the QNLI and SST-2 datasets, following the methodology outlined in (Park and Kim, 2021)[3]. Our analysis reveals that SAFE yields a flatter loss landscape compared to LoRA. This flattening is attributed to SAFE's mechanism of controlling the norm of the weights through regularization effects (See Equation (5)), which consequently enhances resistance to weight perturbations, as depicted in Figure 7(a).

Hessian Eigenvalue Spectrum Analysis. To quantitatively assess the visualized loss landscape shown in Figure 7(a), we perform a detailed analysis of the top-5 Hessian eigenvalue spectrum. A pivotal finding in our analysis is the reduced magnitude of the maximum Hessian eigenvalue, which correlates with a flatter loss landscape, indicative of enhanced generalization potential. Moreover, the diminution of large Hessian eigenvalues facilitates more effective model training (Ghorbani et al., 2019). Furthermore, the suppression of the largest *negative* Hessian eigenvalues markedly contributes to a more convex loss landscape, enhancing the stability of the training process. Figure 7(b) demonstrates that SAFE not only effectively reduces the magnitude of Hessian eigenvalues relative to LoRA but also leads to a smoother and more consistent loss landscape. This evidence highlights the advantages of SAFE in promoting a more reliable and steady training behavior for adapter-tuning.

[3]This involves generating two orthogonal random vectors in a 1D flattened parameter space, which are then normalized and used to perturb the parameters. The strength of these perturbations is determined by their x and y coordinates, with the origin (0, 0) representing the unperturbed state of the parameters, and increasing distance indicating greater perturbation intensity.

Table 4: SAFE improves efficiency over the baseline in all aspects including memory usage, computation amount and training time. Note that we report computation costs for 1-step training.

		NLU	QA	NLG
Memory Usage (GB)	LoRA	20.35	12.75	16.97
	+ SAFE	12.11 40.47%↓	5.85 54.08%↓	11.90 29.91%↓
Computational Cost (TFLOPs)	LoRA	1.24	5.94	8.64
	+ SAFE	0.93 24.74%↓	2.33 60.86%↓	6.79 21.42%↓
Training Time (Normalized)	LoRA	1	1	1
	+ SAFE	0.90 10.29%↓	0.89 10.92%↓	0.80 19.76%↓

5.4 Resource Efficiency

We evaluate the resource efficiency of SAFE in terms of memory usage, computation amount, and training time. Table 4 shows the average resource efficiency improvement for the main results on NLU, QA and NLG tasks. Overall, SAFE reduces memory usage, computation amount, and training time by 42.85%, 34.59%, and 11.82% compared to LoRA, respectively (on average). This means that SAFE can fine-tune twice as many downstream tasks under the same FLOPs budget and further enable on-device fine-tuning for personalization. For example, when fine-tuning a $\text{RoBERTa}_{\text{large}}$ model with a question answering downstream task, SAFE reduces memory usage from 17.73GB to 3.56GB; 8GB is the usual memory size of the edge devices.

5.5 Expanded Experimental Results

Image Classification Task Evaluations. In Appendix A, we conduct comprehensive evaluations of SAFE on a variety of image classification tasks. These experiments consistently demonstrate the efficacy of SAFE, confirming its robust performance across diverse vision-related applications.

Compatibility with Advanced Adapters. Further discussions on the integration of SAFE with various advanced adapter modules are presented in Appendix B. Our results highlight SAFE's versatility and compatibility with multiple adapter-tuning frameworks (Houlsby et al., 2019; Zaken et al., 2022). This adaptability ensures that SAFE's methodology remains effective, independent of specific adapter designs, thereby facilitating scalability across existing adapter-tuning methods.

6 Conclusion

In this paper, we propose SAFE, which selectively freezes adapters for enabling resource efficient fine-tuning of PLMs. We observe that not all adapters contribute equally to adaptation. Motivated by the observation, SAFE gradually freezes less-important adapters, which do not contribute to adaptation during the early training steps. In our evaluation on various models and datasets, SAFE significantly saves memory usage and computation and accelerating training time, with comparable (or even better) accuracy. We also demonstrate that SAFE induces regularization effect, thereby improving generalization performance and accuracy compared to the state-of-the-art PEFT methods. We believe that SAFE can enable resource-efficient fine-tuning of large-scale PLMs, and further pave the path forward to personalized fine-tuning on resource-constrained edge devices.

7 Limitations

We suggest the need for combination with prior research on memory-efficient training. These include low precision, micro-batching, weight sharding, and gradient checkpointing techniques. Though we have not evaluated SAFE along with such memory-efficient training methods, SAFE can be complementarily used along with the methods since SAFE can be applied independently of the training method or weight precision. In particular, since the quantization-based compression technique is quite popular and effective in terms of both compression ratio and preservation of final accuracy, favorable results are expected from combining the proposed technique with the memory-efficient training methods (Han et al., 2015).

Acknowledgments

This work was supported in part by National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2023-00212711), Institute of Information & Communications Technology Planning & Evaluation(IITP)-ITRC(Information Technology Research Center) grant funded by the MSIT(IITP-2025-RS-2023-00260091), ICT Creative Consilience Program through IITP grant funded by the MSIT(IITP-2025-RS-2020-II201819), IITP grant funded by the MSIT (No. RS-2024-00398353, Development of Countermeasure Technologies for Generative AI Security Threats). We also thank the anonymous reviewers for their helpful feedback.

References

Léonard Blier and Yann Ollivier. 2018. The description length of deep learning models. *Advances in Neural Information Processing Systems*, 31.

Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. 2014. Food-101–mining discriminative components with random forests. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI 13*, pages 446–461. Springer.

Junbum Cha, Sanghyuk Chun, Kyungjae Lee, Han-Cheol Cho, Seunghyun Park, Yunsung Lee, and Sungrae Park. 2021. Swad: Domain generalization by seeking flat minima. *Advances in Neural Information Processing Systems*, 34:22405–22418.

Junbum Cha, Kyungjae Lee, Sungrae Park, and Sanghyuk Chun. 2022. Domain generalization by mutual-information regularization with pre-trained models. In *European Conference on Computer Vision*, pages 440–457. Springer.

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

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2023. Longlora: Efficient fine-tuning of long-context large language models. In *The Twelfth International Conference on Learning Representations*.

Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. 2015. The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pages 192–204. PMLR.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR.

Zihao Fu, Haoran Yang, Anthony Man-Cho So, Wai Lam, Lidong Bing, and Nigel Collier. 2023. On the effectiveness of parameter-efficient fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12799–12807.

Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. 2019. An investigation into neural net optimization via hessian eigenvalue density. In *International Conference on Machine Learning*, pages 2232–2241. PMLR.

Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

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

Xuehai He, Chunyuan Li, Pengchuan Zhang, Jianwei Yang, and Xin Eric Wang. 2023. Parameter-efficient model adaptation for vision transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 817–825.

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, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. 2020. Fantastic generalization measures and where to find them. In *International Conference on Learning Representations*.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34:1022–1035.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. 2019. Similarity of neural network representations revisited. In *International conference on machine learning*, pages 3519–3529. PMLR.

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 2013. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 554–561.

Alex Krizhevsky et al. 2009. Learning multiple layers of features from tiny images.

Ya Le and Xuan Yang. 2015. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059.

Sheng Li, Geng Yuan, Yue Dai, Youtao Zhang, Yanzhi Wang, and Xulong Tang. 2022a. Smartfrz: An efficient training framework using attention-based layer freezing. In *The Eleventh International Conference on Learning Representations*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597.

Yuchao Li, Fuli Luo, Chuanqi Tan, Mengdi Wang, Songfang Huang, Shen Li, and Junjie Bai. 2022b. Parameter-efficient sparsity for large language models fine-tuning. *arXiv preprint arXiv:2205.11005*.

Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. 2021. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403.

Baohao Liao, Shaomu Tan, and Christof Monz. 2024. Make pre-trained model reversible: From parameter to memory efficient fine-tuning. *Advances in Neural Information Processing Systems*, 36.

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.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2023. Gpt understands, too. *AI Open*.

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*.

Ilya Loshchilov and Frank Hutter. 2018. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2022. Pointer sentinel mixture models. In *International Conference on Learning Representations*.

Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. 2020. What is being transferred in transfer learning? *Advances in neural information processing systems*, 33:512–523.

Maria-Elena Nilsback and Andrew Zisserman. 2008. Automated flower classification over a large number of classes. In *2008 Sixth Indian conference on computer vision, graphics & image processing*, pages 722–729. IEEE.

Jekaterina Novikova, Ondrej Dusek, and Verena Rieser. 2017. The e2e dataset: New challenges for end-to-end generation. In *18th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 201–206. Association for Computational Linguistics.

Namuk Park and Songkuk Kim. 2021. How do vision transformers work? In *International Conference on Learning Representations*.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. Adapterfusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterhub: A framework for adapting transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 46–54.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR.

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

Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. 2021. Do vision transformers see like convolutional neural networks? *Advances in neural information processing systems*, 34:12116–12128.

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.

Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W Keckler. 2016. vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13. IEEE.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2021. Adapterdrop: On the efficiency of adapters in transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7930–7946.

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*.

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

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Lei Wu and Weijie J Su. 2023. The implicit regularization of dynamical stability in stochastic gradient descent. *arXiv preprint arXiv:2305.17490*.

Minghao Wu, Abdul Waheed, Chiyu Zhang, Muhammad Abdul-Mageed, and Alham Aji. 2024. Lamini-lm: A diverse herd of distilled models from large-scale instructions. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 944–964.

Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Zeke Xie, Fengxiang He, Shaopeng Fu, Issei Sato, Dacheng Tao, and Masashi Sugiyama. 2021. Artificial neural variability for deep learning: On overfitting, noise memorization, and catastrophic forgetting. *Neural computation*, 33(8):2163–2192.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9.

Mingyang Zhang, Hao Chen, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, and Bohan Zhuang. 2023. Loraprune: Pruning meets low-rank parameter-efficient fine-tuning. *arXiv preprint arXiv:2305.18403*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2022. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*.

Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q Weinberger, and Yoav Artzi. Revisiting few-sample bert fine-tuning. In *International Conference on Learning Representations*.

A Results on Image Classification Tasks

We conduct experiments with 8 datasets including class-level transfer and task-level transfer in the image classification tasks within Computer Vision (CV) domain. These datasets include CIFAR-10, CIFAR-100 (Krizhevsky et al., 2009), Country-211 (Radford et al., 2021), Fashion MNIST (Xiao et al., 2017), Food-101 (Bossard et al., 2014), Oxford Flowers (Nilsback and Zisserman, 2008), Standford Cars (Krause et al., 2013) and Tiny ImageNet (Le and Yang, 2015).

Table 5 shows that SAFE can effectively reduce memory usage while achieving comparable accuracy on eight datasets in the image classification task. For example, SAFE achieves a 50.69% memory usage reduction on $\text{ViT}_{\text{large}}$ while maintaining comparable accuracy. Additionally, SAFE remains consistently effective regardless of variations in the pre-trained model size and backbone structure.

Table 5: Experimental results on eight common computer vision tasks. SAFE significantly reduces memory usage while achieving accuracy comparable to the baseline. Note that blue indicates the memory usage reduction rate of SAFE compared to the baseline.

		CIFAR-10	CIFAR-100	Country-211	Fashion MNIST	Food-101	Oxford Flowers	Stanford Cars	Tiny ImageNet	Avg.
ViT_{base}	LoRA	97.77	95.68	16.56	94.52	88.84	99.41	82.56	88.90	**83.03**
	+ SAFE	98.66	95.55	16.29	93.71	88.78	99.61	82.05	89.16	82.98 24.35%↓
$\text{ViT}_{\text{large}}$	LoRA	99.09	96.71	20.44	94.92	90.32	-	86.97	92.13	82.94
	+ LoRA	99.13	97.00	20.44	94.88	90.69	-	86.83	92.03	**83.00** 50.69%↓
$\text{SWIN}_{\text{base}}$	LoRA	98.92	96.20	20.06	95.12	91.36	99.50	87.14	90.31	84.83
	+ SAFE	98.94	96.31	20.26	95.12	91.51	99.71	86.88	90.14	**84.86** 32.01%↓
$\text{SWIN}_{\text{large}}$	LoRA	99.07	97.01	22.19	95.44	92.68	-	85.06	92.02	83.35
	+ LoRA	99.17	96.72	22.66	95.47	92.69	-	85.13	92.11	**83.42** 25.06%↓

B Results with Various PEFT Methods

We validate the applicability of SAFE upon advanced adapter modules (Houlsby et al., 2019; Zaken et al., 2022; Hu et al., 2021). Table 6 shows that SAFE reduces memory usage by 24.76% on average while achieving a comparable GLUE score. This result demonstrates that SAFE can be applied to a variety of adapter-tuning methods to enable resource efficient fine-tuning of large language models.

Table 6: Experimental results for various adapter-tuning methods on the GLUE benchmark. Note that blue indicates the memory usage reduction rate of SAFE compared to the baseline.

$\text{BERT}_{\text{base}}$	CoLA Matthews corr.	SST-2 Accuracy	MNLI Accuracy	RTE Accuracy	QQP Accuracy	MRPC F1 Score	QNLI Accuracy	STS-B Pearson corr.	Avg.
Houlsby	62.38	91.17	83.44	70.50	90.85	89.52	90.59	90.75	83.65
+ SAFE	62.83	93.00	84.15	74.10	90.93	89.70	91.03	91.29	**84.63** 25.70%↓
BitFit	60.92	91.86	82.41	71.94	89.20	88.14	89.80	90.95	**83.15**
+ SAFE	61.73	93.00	81.85	69.78	89.21	88.85	89.51	90.75	83.09 25.53%↓
LoRA	64.46	91.63	82.88	71.22	90.01	88.39	90.01	90.86	83.68
+ SAFE	66.80	90.83	82.03	71.22	89.74	88.51	90.65	90.26	**83.76** 23.06%↓

C Experimental Setup

C.1 Model

We conduct experiments using a pre-trained model deployed on HuggingFace (Wolf et al., 2019). For experiments on the NLU and QA benchmarks, we use bert-base-uncased and bert-large-uncased trained on BookCorpus, a dataset consisting of 11,038 unpublished books and English Wikipedia. We use roberta-base and roberta-large trained on 5 datasets (BookCorpus, English Wikipedia, CC-News, OpenWebText, and Stories) for the RoBERTa model. For experiments on the NLG benchmark, we use GPT-2 medium and GPT-2 large distributed by OpenAI. We also employ the LLaMA-2 7B model, an open-weight large language model released by Meta, which has been trained on a mixture of publicly available and licensed datasets, including Common Crawl, C4, GitHub, Wikipedia, Project Gutenberg, ArXiv, and Stack Exchange. We use vit-base-patch16-224-in21k and vit-large-patch16-224-in21k distributed by Google for experiments in the ViT model. Finally, We use the swin-base-patch4-window7-224 and swin-large-patch4-window7-224 models distributed by Microsoft for experiments in the SWIN model.

C.2 Computing Resources

Our experimental setup leverages 2 RTX4090 with 24GB memory for NLU, QA, and NLG tasks and 1 RTX 4090 for CV downstream task, excluding the LLaMA experiments.

C.3 Dataset Statistics

We present the dataset statistics of GLUE (Wang et al., 2018), SQuAD (Rajpurkar et al., 2016), E2E NLG Challenge (Novikova et al., 2017), and WikiText-2 (Merity et al., 2022) in following table. We fine-tune models on the LaMini instruction dataset (Wu et al., 2024) for LLaMA-2_{7B} and evaluate their performance on WikiText-2 using perplexity.

Table 7: Summary of the NLU, QA, and NLG benchmarks.

NLU Benchmark						
Dataset	# Train	# Valid	# Test	# Label	Task	Evaluation Metric
Single-Sentence Classification (GLUE)						
CoLA	8,551	521	522	2	Acceptability	Matthews corr
SST-2	66,349	1,000	872	2	Sentiment	Accuracy
Pairwise Text Classification (GLUE)						
MNLI	392,702	9,832	9,815	3	NLI	Accuracy
RTE	2,490	138	139	2	NLI	Accuracy
QQP	362,846	1,000	40,431	2	Paraphrase	Accuracy
MRPC	3,668	204	204	2	Paraphrase	F1 score
QNLI	103,743	1,000	5,463	2	QA/NLI	Accuracy
Pairwise Text Classification (GLUE)						
STS-B	5,749	750	750	1	Similarity	Pearson corr
QA Benchmark						
Dataset	# Train	# Valid	# Test	# Label	Task	Evaluation Metric
SQuAD	87,600	5,300	5,300	2	Question Answering	F1 score
NLG Benchmark						
Dataset	# Train	# Valid	# Test	# Label	Task	Evaluation Metric
E2E NLG Challenge	42,061	4,672	4,693		Generation	BLEU, NIST, METEOR, ROUGE-L, and CIDEr
WikiText-2	2,589,000	1,000	4,360		Language Modeling	Perplexity (PPL)

The following table lists dataset statistics evaluated in the CV domain.

Table 8: Summary of CV benchmark.

CV Benchmark						
Dataset	# Train	# Valid	# Test	# Label	Task	Evaluation Metric
CIFAR-10	45,000	5,000	10,000	10	Classification	Accuracy
CIFAR-100	45,000	5,000	10,000	100	Classification	Accuracy
Fashion MNIST	54,000	6,000	10,000	10	Classification	Accuracy
Oxford Flowers	6,453	717	1,020	102	Classification	Accuracy
Food-101	68,220	7,580	25,300	102	Classification	Accuracy
Country-211	25,920	2,880	21,100	211	Classification	Accuracy
Stanford Cars	7,326	814	8,040	196	Classification	Accuracy
Tiny ImageNet	90,000	10,000	10,000	200	Classification	Accuracy

C.4 Hyperparameter Settings

We explore 10% of all epochs for at least 5 learning rates. Hyperparameter settings, including learning rate, are made by referring to previous works (He et al., 2023; Houlsby et al., 2019; Hu et al., 2021; Zaken et al., 2022). We use the AdamW optimizer (Loshchilov and Hutter, 2018) and LinearLR learning rate scheduler and set weight decay to 0 in experiments. In our evaluation, we configure LoRA as follows: r = 4, alpha = 16, target modules = ["query", "value"], and LoRA dropout = 0.1.

Since the lower layers of the BERT model are more easily converted to fine-tuning than the upper layers, we extend the previous study (Zhang et al.) that froze most of the lower layers with adapter-tuning and experiment with a baseline that naively inserts adapters only into six layers. For LoRAPrune, we configure the pruning ratio to 0.5, pruning frequency to 10, and the pruning metric to 'lora,' as these settings have shown effective results in prior work (Zhang et al., 2023). In the case of MEFT, for a fair comparison, we set the reversible layers to half of the total number of layers. We employ the MEFT_1 method, where the $\mathcal{F}$ architecture is the layer, from the three reversible transformation methods proposed in MEFT. For the hyperparameters of MEFT, such as λ and β, we follow the values proposed in previous work (Liao et al., 2024) to ensure consistency in the evaluation.

Table 9: Hyperparameter settings on the NLU and QA tasks.

pre-trained model	dataset	method	final learning rate	batch size	# epochs
BERT-base-uncased	GLUE / CoLA	LoRA, LoRA + SAFE	6.00E-04	32	100
		BitFit, BitFit + SAFE	9.00E-04	32	100
		Houlsby, Houlsby + SAFE	5.00E-04	32	100
	GLUE / SST-2	LoRA, LoRA + SAFE	7.00E-04	32	75
		BitFit, BitFit + SAFE	7.00E-04	32	75
		Houlsby, Houlsby + SAFE	2.00E-04	32	75
	GLUE / MNLI	LoRA, LoRA + SAFE	9.00E-04	32	50
		BitFit, BitFit + SAFE	8.00E-04	32	50
		Houlsby, Houlsby + SAFE	4.00E-04	32	50
	GLUE / RTE	LoRA, LoRA + SAFE	9.00E-04	32	100
		BitFit, BitFit + SAFE	8.00E-04	32	100
		Houlsby, Houlsby + SAFE	4.00E-04	32	100
	GLUE / QQP	LoRA, LoRA + SAFE	4.00E-04	32	50
		BitFit, BitFit + SAFE	6.00E-04	32	50
		Houlsby, Houlsby + SAFE	4.00E-04	32	50
	GLUE / MRPC	LoRA, LoRA + SAFE	5.00E-04	16	50
		BitFit, BitFit + SAFE	5.00E-04	32	50
		Houlsby, Houlsby + SAFE	5.00E-04	32	50
	GLUE / QNLI	LoRA, LoRA + SAFE	5.00E-04	32	50
		BitFit, BitFit + SAFE	5.00E-04	32	50
		Houlsby, Houlsby + SAFE	4.00E-04	32	50
	GLUE / STS-B	LoRA, LoRA + SAFE	8.00E-04	32	50
		BitFit, BitFit + SAFE	9.00E-04	32	50
		Houlsby, Houlsby + SAFE	5.00E-04	32	50
	SQuAD	LoRA, LoRA + SAFE	3.00E-04	16	50
		BitFit, BitFit + SAFE	9.00E-04	16	50
		Houlsby, Houlsby + SAFE	1.00E-04	16	50
BERT-large-uncased	GLUE / CoLA	LoRA, LoRA + SAFE	1.00E-04	32	80
	GLUE / SST-2	LoRA, LoRA + SAFE	6.00E-04	32	60
	GLUE / MNLI	LoRA, LoRA + SAFE	1.00E-04	16	40
	GLUE / RTE	LoRA, LoRA + SAFE	6.00E-04	32	80
	GLUE / QQP	LoRA, LoRA + SAFE	3.00E-04	16	40
	GLUE / MRPC	LoRA, LoRA + SAFE	3.00E-04	4	50
	GLUE / QNLI	LoRA, LoRA + SAFE	2.00E-04	8	50
	GLUE / STS-B	LoRA, LoRA + SAFE	8.00E-04	32	50
		Full-param Fine-tuning	7.00E-05	16	50
	SQuAD	LoRA, LoRA + SAFE	3.00E-04	16	50
		BitFit, BitFit + SAFE	9.00E-04	16	50
		Houlsby, Houlsby + SAFE	1.00E-04	16	50
		Pfeiffer, Pfeiffer + SAFE	3.00E-04	16	50
		AdaLoRA, AdaLoRA + SAFE	4.00E-04	16	50
RoBERTa-base	SQuAD	LoRA, LoRA + SAFE	5.00E-04	32	50
		BitFit, BitFit+SAFE	8.00E-04	32	50
		Houlsby, Houlsby+SAFE	4.00E-04	32	50
RoBERTa-large	SQuAD	LoRA, LoRA + SAFE	6.00E-04	16	50
		BitFit, BitFit+SAFE	7.00E-04	16	50
		Houlsby, Houlsby+SAFE	4.00E-04	16	50

Table 10: Hyperparameter settings on the NLG task.

pre-trained model	**GPT-2 medium**	**GPT-2 large**	**LLaMA-2 7B**
Training			
final learning rate	1.00E-04	5.00E-05	1.00E-04
batch size	8	4	128 (w/ micro batch size 2)
# epochs	10	10	3
Seq Length	512	512	512
Label Smooth	0.1	0.1	
Inference			
Beam Size	10	10	
Length Penalty	0.8	0.8	
no repeat ngram size	4	4	

Table 11: Hyperparameter settings on the CV task.

pre-trained model	dataset	method	final learning rate	batch size	# epochs
ViT-base-patch16-224	CIFAR-10	BitFit, BitFit + SAFE	3.00E-03	64	100
		LoRA, LoRA + SAFE	3.00E-03	64	100
	CIFAR-100	BitFit, BitFit + SAFE	3.00E-03	64	100
		LoRA, LoRA + SAFE	3.00E-03	64	100
	Fashion MNIST	BitFit, BitFit + SAFE	4.00E-03	64	100
		LoRA, LoRA + SAFE	3.00E-03	64	100
	Oxford Flowers	BitFit, BitFit + SAFE	2.00E-03	64	30
		LoRA, LoRA + SAFE	8.00E-04	64	40
	Food-101	BitFit, BitFit + SAFE	4.00E-03	64	100
		LoRA, LoRA + SAFE	3.00E-03	64	100
	Tiny ImageNet	BitFit, BitFit + SAFE	1.00E-03	64	100
		LoRA, LoRA + SAFE	8.00E-04	64	100
	Country-211	BitFit, BitFit + SAFE	2.00E-03	64	100
		LoRA, LoRA + SAFE	4.00E-03	64	100
	Stanford Cars	BitFit, BitFit + SAFE	9.00E-03	64	100
		LoRA, LoRA + SAFE	7.00E-03	64	100
ViT-large-patch16-224	CIFAR-10	BitFit, BitFit + SAFE	4.00E-03	64	100
		LoRA, LoRA + SAFE	6.00E-04	64	100
	CIFAR-100	BitFit, BitFit + SAFE	2.00E-03	64	100
		LoRA, LoRA + SAFE	5.00E-04	64	100
	Fashion MNIST	BitFit, BitFit + SAFE	3.00E-03	64	100
		LoRA, LoRA + SAFE	9.00E-04	64	100
	Oxford Flowers	BitFit, BitFit + SAFE	-	-	-
		LoRA, LoRA + SAFE	-	-	-
	Food-101	BitFit, BitFit + SAFE	9.00E-04	64	100
		LoRA, LoRA + SAFE	7.00E-04	64	100
	Tiny ImageNet	BitFit, BitFit + SAFE	8.00E-04	64	100
		LoRA, LoRA + SAFE	6.00E-04	64	100
	Country-211	BitFit, BitFit + SAFE	2.00E-03	64	100
		LoRA, LoRA + SAFE	9.00E-04	64	100
	Stanford Cars	BitFit, BitFit + SAFE	1.00E-03	64	100
		LoRA, LoRA + SAFE	1.00E-03	64	100
SWIN-base-patch4-window7-224	CIFAR-10	LoRA, LoRA + SAFE	1.00E-03	64	50
	CIFAR-100	LoRA, LoRA + SAFE	1.00E-03	64	50
	Fashion MNIST	LoRA, LoRA + SAFE	1.00E-03	64	50
	Oxford Flowers	LoRA, LoRA + SAFE	7.00E-04	64	30
	Food-101	LoRA, LoRA + SAFE	9.00E-04	64	50
	Tiny ImageNet	LoRA, LoRA + SAFE	1.00E-03	64	50
	Country-211	LoRA, LoRA + SAFE	7.00E-04	64	50
	Stanford Cars	LoRA, LoRA + SAFE	1.00E-03	64	50
SWIN-large-patch4-window7-224	CIFAR-10	LoRA, LoRA + SAFE	8.00E-04	64	50
	CIFAR-100	LoRA, LoRA + SAFE	7.00E-04	64	50
	Fashion MNIST	LoRA, LoRA + SAFE	1.00E-03	64	50
	Oxford Flowers	LoRA, LoRA + SAFE	-	-	-
	Food-101	LoRA, LoRA + SAFE	5.00E-04	64	50
	Tiny ImageNet	LoRA, LoRA + SAFE	6.00E-04	64	50
	Country-211	LoRA, LoRA + SAFE	6.00E-04	64	50
	Stanford Cars	LoRA, LoRA + SAFE	3.00E-03	64	50