Flexora: Flexible Low-Rank Adaptation for Large Language Models

Chenxing Wei[*†§], **Yao Shu**[*§], **Ying Tiffany He**[†], **Fei Richard Yu**[#†‡]
[†]College of Computer Science and Software Engineering, Shenzhen University, China
[§]Guangdong Lab of AI and Digital Economy (SZ), China
[‡]School of Information Technology, Carleton University, Canada
weichenxing2023@email.szu.edu.cn, shuyao@gml.ac.cn
heying@szu.edu.cn, richard.yu@ieee.org

Abstract

Large language models (LLMs) have revolutionized artificial intelligence, but their performance on specific tasks is often limited by knowledge boundaries. While fine-tuning techniques like low-rank adaptation (LoRA) aim to address this, they can suffer from overfitting. We propose *flexible low-rank adaptation* (Flexora), a novel method that automatically selects the most critical layers for fine-tuning to optimize performance across diverse downstream tasks. Flexora formulates layer selection as a hyperparameter optimization problem, employs unrolled differentiation for efficient solving, and identifies the most impactful layers based on optimized hyperparameters. Extensive experiments across various pre-trained models and natural language tasks demonstrate that Flexora consistently outperforms existing baselines. We provide theoretical insights and comprehensive ablation studies to elucidate the effectiveness of Flexora. Therefore, Flexora offers a robust solution to enhance LoRA fine-tuning for LLMs, potentially advancing the field of adaptive language model optimization.

1 Introduction

The advent of large language models (LLMs) [1, 2] has revolutionized artificial intelligence, offering unprecedented capabilities across various domains. However, this progress comes at a significant cost: LLMs demand substantial computational resources due to their vast parameter sets and complex functionalities [3, 4]. This challenge has spurred the development of parameter-efficient fine-tuning (PEFT) methods [5, 6], with low-rank adaptation (LoRA) [7] emerging as a particularly promising approach. LoRA's innovation lies in its ability to freeze pre-trained parameters while introducing trainable auxiliary parameters ($\Delta \boldsymbol{W}$) at each layer, dramatically reducing training costs while maintaining impressive performance. However, despite its widespread adoption, LoRA is not without limitations. It can underperform on certain tasks, likely due to overfitting issues, as evidenced in benchmarks like GLUE [8], summary tasks [9], and complex reasoning tasks [10]. Existing techniques to combat overfitting, such as dropout [11] and novel regularization strategies [12], often yield performance comparable to or lower than vanilla LoRA and lack the flexibility to adapt across different tasks. Moreover, current methods typically require manual hyperparameter tuning, limiting their practical applicability in diverse scenarios. These challenges therefore underscore the urgent need for an algorithm that delivers superior performance, enables automatic hyperparameter tuning, and supports flexible training across various tasks.

To address these limitations, we introduce *flexible low-rank adaptation* (Flexora), a novel framework designed to flexibly fine-tune LLMs using an automated layer-level policy. Our approach is inspired

$*$ Equal contribution, $\#$ corresponding author.

38th Workshop on Fine-Tuning in Machine Learning (NeurIPS 2024).

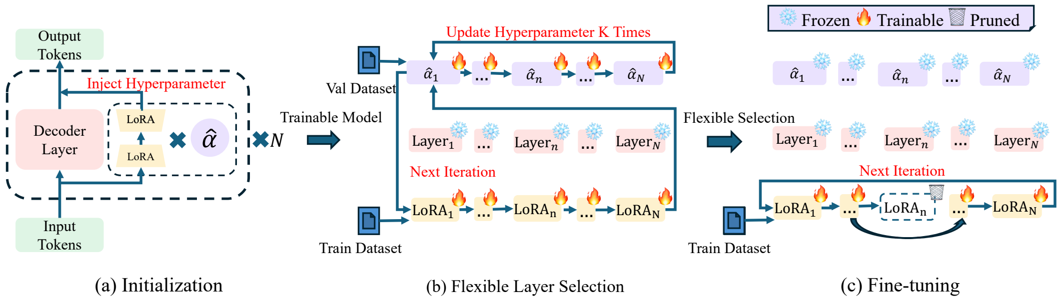

Figure 1: An overview of Flexora: (a) Initialization of hyperparameters $\widehat{\alpha}$ and their integration with LoRA parameters to produce the Trainable Model. (b) Simultaneous training of LoRA parameters and hyperparameters $\widehat{\alpha}$ using different datasets, minimizing empirical risk for both validation and training datasets. The hyperparameter vector $\widehat{\alpha}$ is then ranked based on magnitude. (c) Flexible selection of layers to be trained, where higher-ranked layers are activated for training while others remain frozen.

by hyperparameter optimization (HPO) and offers several key innovations. First, we demonstrate that fine-tuning only the most critical layers can significantly reduce overfitting and enhance performance. Second, we frame the layer selection problem as an HPO task and employ unrolled differentiation (UD) to solve it efficiently. Third, we develop a three-stage process that automatically identifies and focuses on the most important layers for downstream tasks. As illustrated in Figure 1, Flexora operates through an initialization stage (Sec. 4.1) that injects defined hyperparameters into LoRA parameters, a flexible layer selection stage (Sec. 4.2) that optimizes these hyperparameters using UD, and a fine-tuning stage (Sec. 4.3) that selectively updates only the most crucial layers, significantly reducing computational overhead. Our extensive empirical results (Sec. 5) demonstrate that Flexora effectively reduces unimportant LoRA parameters, mitigates overfitting, and enhances overall performance across a variety of tasks and model architectures.

In summary, our key contributions consist of: (a) the introduction of Flexora, a novel framework for automatic layer selection in LoRA fine-tuning; (b) a formulation of layer selection as an HPO task, efficiently solved using unrolled differentiation; (c) comprehensive validation through extensive experiments on various LLMs and downstream tasks; and (d) theoretical insights into the performance improvements achieved by Flexora, providing a deeper understanding of its effectiveness.

2 Related Work

Low-Rank Adaptation (LoRA) Low-rank adaptation (LoRA) methods are widely used to reduce training parameters when fine-tuning large language models (LLMs) for specific applications. However, LoRA often suffers from overfitting, which can degrade performance on downstream tasks. To mitigate this, various strategies have been proposed: LoRA-SP [8] randomly freezes half of the LoRA parameters during fine-tuning to alleviate overfitting; LoRA-FA [13] freezes down-projection weights while updating only up-projection weights; LoRA-drop [14] prunes less important parameters based on layer output analysis; LoRAPrune [10] jointly prunes parts of the LoRA matrix and LLM parameters based on gradients; and LoRAShear [15] employs knowledge-based structured pruning to reduce computational costs while enhancing generalization. Despite their benefits, these methods often *(a)* require significant design effort, *(b)* struggle to adapt across different tasks, and *(c)* can be overly complex for practical application. In contrast, we introduce Flexora, a framework designed for flexible LoRA fine-tuning across various tasks using a simple, automated layer-level policy.

Hyperparameter Optimization (HPO) HPO is widely applied across various domains. Specifically, in the domain of neural architecture search, DARTS [16] conceptualizes the coefficients defining the network architecture as hyperparameters. In the domains of feature learning, DS^{3}L [17] considers feature extractors as hyperparameters. In the field of data science, TPOT [18] employs hyperparameters as weights to measure the importance of data. By minimizing the validation loss over these hyperparameters, the optimal variables, e.g., the architectures in [16], the features in [17], and the data in [18], are identified, leading to superior performance in their respective domains. Drawing inspiration from these works, we initially formulated the layer selection in the LoRA method as an

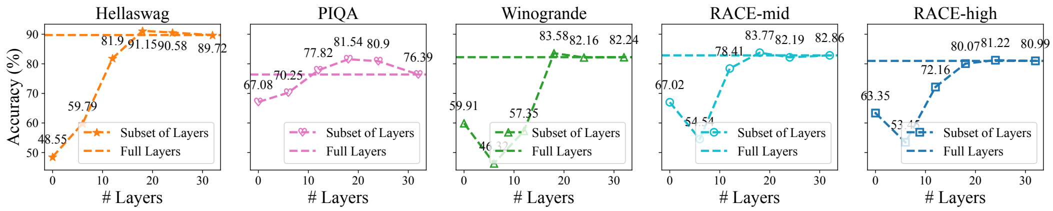

Figure 2: This figure depicts the relationship between the number of fine-tuning layers and model accuracy across four distinct datasets: Hellaswag, PIQA, Winogrande, and RACE, with the latter including two separate tasks, RACE-mid and RACE-high, which vary in difficulty. The x-axis represents the number of fine-tuned layers, ranging from 0 to 32, where 0 corresponds to the base model without fine-tuning. Selected configurations include 6, 12, 18, 24, and 32 randomly fine-tuned layers. The full 32-layer configuration, representing the vanilla LoRA setup, is shown as a horizontal dashed line in the plots. The y-axis indicates model accuracy as a percentage.

HPO problem. This involves optimizing hyperparameters to quantify the contributions of different layers, aiming to achieve optimal performance on downstream tasks and thereby select the most crucial layers for fine-tuning. This formulation subsequently led to the development of our Flexora.

3 Preliminaries

In this section, we first provide empirical insights showing that layer selection is crucial for improving the performance of LLMs in Sec. 3.1, and then frame the layer selection problem as a well-defined HPO problem in Sec. 3.2.

3.1 Empirical Insights

To study the impact of the number of LoRA fine-tuning layers on overall performance, we conducted a preliminary study using Llama3-8B [19] across a range of downstream tasks. Here, we randomly selected different subsets of layers for LoRA fine-tuning and assessed their performance on these tasks. The findings, shown in Figure 2, reveal a clear trend: while increasing the number of fine-tuned layers generally improves model performance, there is a critical point beyond which fine-tuning more layers leads to potential overfitting and subsequent performance decline. This hence suggests that selecting an optimal subset of layers for LoRA fine-tuning is crucial for maximizing performance, which interestingly aligns with the previous empirical studies[20, 14, 15].

3.2 Problem Formulation

Inspired by the empirical insights above, we aim to identify the most critical layers in LoRA fine-tuning to improve generalization performance across a variety of downstream tasks. Formally, we consider an N-layer LLM with LoRA fine-tuning parameters $\theta \in \mathbb{R}^d$, and let the hyperparameter $\alpha \in \{0,1\}^N$ denote the selection of fine-tuning layers, where a value of 1 indicates that a layer is selected for fine-tuning. Given the test data distribution $\mathcal{D}_{\text{test}}$ and the training dataset S_{train}, we then define the expected test and training error as $\mathcal{R}^{\text{test}}(\theta,\alpha) \triangleq \mathbb{E}_{x\sim\mathcal{D}_{\text{test}}}[\ell(x,\theta;\alpha)]$ and $\mathcal{R}^{\text{train}}(\theta,\alpha) \triangleq \mathbb{E}_{x\sim S_{\text{train}}}[\ell(x,\theta;\alpha)]$, respectively. Hence, to select the optimal LoRA fine-tuning layers for maximized performance on downstream tasks, we aim to solve the following bilevel optimization problem:

$$\min_{\alpha\in\{0,1\}^N} \mathcal{R}^{\text{test}}(\theta^*(\alpha),\alpha) \quad \text{s.t.} \quad \theta^*(\alpha) = \arg\min_{\theta\in\mathbb{R}^d} \mathcal{R}^{\text{train}}(\theta,\alpha)\,. \tag{1}$$

This formulation follows a standard hyperparameter optimization (HPO) approach as demonstrated in [21], where α serves as the hyperparameter. Thus, the layer selection problem for LoRA fine-tuning in LLMs is framed as a well-defined HPO problem.

Unfortunately, it is typically infeasible to access the full test distributions, denoted by $\mathcal{D}_{\text{test}}$, for this optimization. This expected test error can typically be approximated by the empirical validation error based on the validation dataset S_{val}, which is defined as $\widehat{\mathcal{R}}^{\text{val}}(\theta,\alpha) \triangleq \mathbb{E}_{x\sim S_{\text{val}}}[\ell(x,\theta;\alpha)]$ Therefore, (1) can be simplified as:

$$\min_{\alpha\in\{0,1\}^N} \widehat{\mathcal{R}}^{\text{val}}(\theta^*(\alpha),\alpha) \quad \text{s.t.} \quad \theta^*(\alpha) = \arg\min_{\theta\in\mathbb{R}^d} \mathcal{R}^{\text{train}}(\theta,\alpha)\,. \tag{2}$$

4 The Flexora Framework

To address the layer selection problem defined above, we propose our *flexible low-rank adaptation for LLMs* (Flexora) framework in Figure 1. As illustrated in Figure 1, the Flexora framework consists of three key stages: an initial stage (detailed in Sec. 4.1), a flexible layer selection stage (detailed in Sec. 4.2), and a fine-tuning stage for the selected LoRA layers (detailed in Sec. 4.3).

4.1 Initial Stage

We begin by introducing a special formulation of LoRA, which incorporates the layer selection hyperparameter $\alpha = (\alpha^{(1)}, \cdots, \alpha^{(N)}) \in \{0, 1\}^N$, as follows:

$$h^{(i)} = Wz^{(i)} + \alpha^{(i)} B^{(i)} A^{(i)} z^{(i)}, \quad \text{s.t.} \quad \alpha_i \in \{0, 1\} \,. \tag{3}$$

Here, $h^{(i)}$ is the output of the i-th layer, where W is the original weight matrix, $z^{(i)}$ is the input, and $B^{(i)}$ and $A^{(i)}$ are the low-rank adaptation matrices of LoRA. The hyperparameter $\alpha^{(i)}$ determines whether LoRA is applied for layer i. Specifically, if $\alpha^{(i)} = 0$, the equation simplifies to $h^{(i)} = Wz^{(i)}$, meaning the i-th layer reverts to standard computation without LoRA, implying that the additional complexity of LoRA is unnecessary for layer i. Conversely, when $\alpha^{(i)} = 1$, the equation becomes the standard LoRA form, $h^{(i)} = Wz^{(i)} + B^{(i)} A^{(i)} z^{(i)}$, indicating that LoRA significantly enhances the performance of layer i by allowing the low-rank matrices to better capture complex patterns. So, this dynamic adjustment allows the model to selectively apply LoRA when a specific layer is most beneficial, thereby optimizing the fine-tuning process and mitigating the risk of overfitting.

However, due to the inherent difficulty of directly optimizing the discrete layer selection hyperparameter α, we adopt a continuous relaxation approach by replacing the α in (3) with its continuous counterpart, $\widehat{\alpha} = (\widehat{\alpha}^{(1)}, \cdots, \widehat{\alpha}^{(N)})$:

$$h^{(i)} = W_0 z^{(i)} + \widehat{\alpha}^{(i)} B^{(i)} A^{(i)} z^{(i)}, \quad \text{s.t.} \quad \widehat{\alpha}^{(i)} = \frac{\exp\left(\alpha^{(i)}\right)}{\sum_{i \in [N]} \exp\left(\alpha^{(i)}\right)} N \,. \tag{4}$$

Notably, $\alpha \in \mathbb{R}^N$ now and α are typically initialized to zeros, providing a neutral starting point where no layer is initially excluded from LoRA fine-tuning. Meanwhile, the constant scale N ensures that when all layers are selected for fine-tuning, the scale of each selected layer for LoRA fine-tuning is preserved, resulting in $\widehat{\alpha}^{(i)} = 1$ for all layers, aligning with the vanilla LoRA scale as shown above.

4.2 Flexible Layer Selection Stage

Optimization Strategy. Given the continuous relaxation $\widehat{\alpha}$ defined above, we propose to solve the well-defined HPO problem in Equation 2 using the widely applied unrolled differentiation (UD) method [22–26]. The UD method typically involves two alternating optimization processes: (a) the inner-level and (b) the outer-level optimization. In this paper, the inner-level optimization is defined as $\arg\min_{\theta \in \mathbb{R}^d} \mathcal{R}^{\text{train}}(\theta, \alpha)$, in which the layer selection hyperparameter α is fixed, and the LoRA parameters θ are updated using stochastic gradient methods (e.g., SGD [27] or AdamW [28]) on the training dataset S_{train}. This step focuses on optimizing model performance by adjusting the parameters associated with the selected layers (line 3 in Algorithm 1). Meanwhile, the outer-level optimization is $\arg\min_{\alpha \in \mathbb{R}^N} \widehat{\mathcal{R}}^{\text{val}}(\theta, \alpha)$, in which the layer selection hyperparameter α is updated using stochastic gradient methods (e.g., SGD) based on the validation performance of the optimized LoRA parameters θ from the inner-level process (lines 4–9 in Algorithm 1). This step intends to maximize the validation performance of LoRA fine-tuning based on a subset of selected layers. These two alternating processes therefore iteratively refine both the model parameters and the layer selection criteria, making LoRA layer selection more computationally efficient in practice. After T iterations of these alternating processes, the final iteration α_T is output as the optimal layer selection denoted as α^* (line 10 in Algorithm 1).

Selection Strategy. To begin with, we introduce the following proposition:

Proposition 1. *If α is initialized to zeros, then for any $T \geq 0$ and $K \geq 0$ in Alg. 1, $\sum_{i=1}^{N} \alpha^{(i)} = 0$.*

The proof of this proposition is provided in Appendix A.1. This result highlights that the mean value of the hyperparameter α remains 0, indicating that after the layer selection stage, the elements in the

Algorithm 1 The Flexora Framework

Input: Number of outer-level iteration K; Number of alternating iteration T; Initialized LoRA parameters θ_0 and hyperparameter $\alpha_0 = 0$; Learning rate η_α and η_θ
for $t = 0$ **to** $T - 1$ **do**
  $\theta_{t+1} \leftarrow \theta_t - \eta_\theta \nabla_\theta \mathcal{R}^{\text{train}}(\theta, \alpha_t)|_{\theta=\theta_t}$
  $\alpha_{t+1,0} \leftarrow \alpha_t$
  for $k = 0$ **to** $K - 1$ **do**
    $\alpha_{t+1,k+1} \leftarrow \alpha_{t+1,k} - \eta_\alpha \nabla_\alpha \widehat{\mathcal{R}}^{\text{val}}(\theta_{t+1}, \alpha)|_{\alpha=\alpha_{t+1,k}}$
  end for
  $\alpha_{t+1} \leftarrow \alpha_{t+1,K}$
end for
return $\alpha^* = \alpha_T$

optimized hyperparameter α^* can take on both positive and negative values. We therefore propose to determine the layers for LoRA fine-tuning by selecting those where $\alpha^{(i)} > 0$. That is, we focus on layers where $\widehat{\alpha}^{(i)} > 1$ according to (4), as these layers are expected to contribute positively to the final fine-tuning performance compared to the baseline scale of $\widehat{\alpha}^{(i)} = 1$ in the vanilla LoRA. In contrast, layers where $\widehat{\alpha}^{(i)} < 1$ are considered less beneficial for LoRA fine-tuning. As a result, this method not only facilitates automatic layer selection but also provides flexibility in adjusting the number and specific layers for LoRA fine-tuning, helping to mitigate the potential overfitting and improve overall performance in LLM fine-tuning.

4.3 Fine-Tuning Stage

During the fine-tuning stage, as illustrated in Figure 1c, we adopt a selective activation strategy. In this phase, we freeze the layers not selected for fine-tuning, keeping their parameters unchanged, and focus on retraining only the selected layers to enhance performance. This targeted approach concentrates computational resources on the most critical layers for the downstream task. By retraining the LoRA parameters from scratch in these layers, the model adaptively learns optimal representations, reducing the risk of overfitting and improving performance, especially for simpler tasks. We will validate this approach with the empirical results presented below.

5 Empirical Results

In this section, we present comprehensive experiments to support the effectiveness and efficiency of our Flexora framework with datasets and experimental setup detailed in Sec. 5.1, main results detailed in Sec. 5.2, and ablation studies detailed in Sec. 5.3.

5.1 Datasets and Setup

To evaluate the performance of our proposed Flexora method, we utilize benchmark datasets such as Winogrande, RACE, PIQA, and Hellaswag, which cover various reasoning and comprehension tasks, using accuracy as the metric. Winogrande [29] assesses commonsense reasoning with 44,000 questions, RACE [30] focuses on reading comprehension with nearly 100,000 questions, PIQA [31] evaluates physics commonsense reasoning with over 16,000 Q&A pairs, and Hellaswag [32] tests commonsense natural language reasoning with 70,000 questions. Our experimental setup includes 11 mainstream large-scale language models (LLMs), such as Llama3-8B [19], Chatglm3-6B [33], Mistral-7B-v0.1 [34], and Gemma-7B [35]. Our Flexora is implemented on the Llama-factory [36] framework and evaluated with the opencompass [37] framework. The baselines include pre-trained models, Full FT, LoRA, AdaLoRA, LoRA-drop, LoRAShear, and others. The experimental setup is detailed in Appendix B. All experiments are conducted on a single NVIDIA A100 GPU.

5.2 Main Results

In this section, we evaluate the performance improvement and training efficiency of Flexora on Llama3-8B, and the results are listed in Table 1 (Accuracy) and Table 2 (Time and Parameter).

Table 1: Comparison of accuracy across various common sense reasoning tasks using Llama3-8B. The baseline experimental configuration is detailed in Appendix B. Here, "Pre-trained" refers to using the base model for reasoning, "Full FT" indicates full parameter fine-tuning, and "Random (Greedy)" represents the best result from randomly selected layers.

Methods	Hellaswag	PIQA	Winogrande	RACE-mid	RACE-high	Average
Pre-trained	48.55	67.08	59.91	67.02	63.35	61.18
Full FT	90.53	79.32	81.16	81.92	79.36	82.46
LLaMA Pro	90.03	76.23	80.32	82.68	80.96	82.04
LoRA	89.72	76.39	82.24	82.86	80.99	83.04
OLoRA	91.32	80.69	83.01	83.65	77.66	83.27
LoRA+	90.62	77.92	83.06	83.69	81.80	83.42
PiSSA	91.33	77.77	83.72	84.35	82.15	83.86
VeRA	90.98	78.63	83.64	83.55	78.84	83.13
LoRAPrune (Ratio = 0.5)	88.42	77.12	81.23	82.96	80.42	82.03
AdaLoRA ($r_0 = 4$)	90.17	80.20	77.19	83.15	77.93	81.73
LoRA-drop	91.86	77.91	76.46	77.30	75.24	79.75
Random (Greedy)	91.15	81.54	83.58	83.77	81.22	84.25
Flexora	**93.62**	**85.91**	**85.79**	**84.61**	**82.36**	**86.46**

Table 2: Comparison of training time and parameters, with the green font indicating the reduction ratio, is conducted on a single NVIDIA A100 GPU using Llama3-8B. The time metric reflects the wallclock time for the fine-tuning phase of LoRA and Flexora, excluding the layer selection phase.

Metrics	Method	Hellaswag	PIQA	Winogrande	RACE
Time (h)	LoRA	5.30	4.03	4.96	8.37
	Flexora	**4.71 (11.1%)**	**3.87 (4.0%)**	**3.84 (22.6%)**	**7.46 (10.9%)**
# Params (M)	LoRA	3.4	3.4	3.4	3.4
	Flexora	**2.00 (41.2%)**	**1.70 (50.0%)**	**1.70 (50.0%)**	**1.70 (50.0%)**

Specifically, the experimental results using LoRAShear are detailed in Appendix C.3, while the loss metrics are discussed in Appendix D.1. The results show that Flexora outperforms all baseline methods. Specifically, compared with full fine-tuning, Flexora only fine-tunes 0.02% of the parameters while achieving superior performance. Compared with LoRA and enhanced LoRA methods, Flexora fine-tunes about half of the parameters of these methods, but still outperforms all methods. This shows that fine-tuning too many parameters will lead to overfitting, which not only fails to improve the model performance on downstream tasks, but may also degrade the generalization of the model due to the overfitting effect. Therefore, it is crucial to select the most relevant layers in the downstream tasks for optimization. Flexible layer selection stage of Flexora is able to consider the relationship between the pre-trained parameters of each LLM layer and the downstream tasks. This stage effectively identifies the most critical layers for various downstream tasks, and minimizes the risk of model overfitting by focusing on training these layers, thereby achieving excellent performance. We also discuss the impact of different search samples on the training time and final performance of our Flexible layer selection stage, detailed in Appendix C.4. In particular, Flexora surpasses LoRA-drop, which always selects the last layers, by accurately identifying the most important layers. In addition, Flexora shows strong generalization and scalability across different LLMs. As shown in Figure 3 and detailed in Appendix C.1, almost all LLMs can leverage Flexora to significantly improve performance with fewer fine-tuned parameters. In Appendix E, we compared Flexora with LoRA using specific examples. The model fine-tuned with Flexora outperforms LoRA on challenging cases and provides correct explanations for answers not seen in the training set, showcasing its robust learning and generalization capabilities.

5.3 Ablation Studies

Effective Layer Selection in Flexora. In the first ablation experiment, we maintained the number of layers selected by Flexora unchanged but chose different layers for fine-tuning, aiming to verify whether Flexora selected the right layers. The experimental results are shown in Table 3. The result underscores two key points: First, Flexora can precisely determine the number of layers for fine-tuning. Even when the specific fine-tuning layers are chosen at random, the results continue to

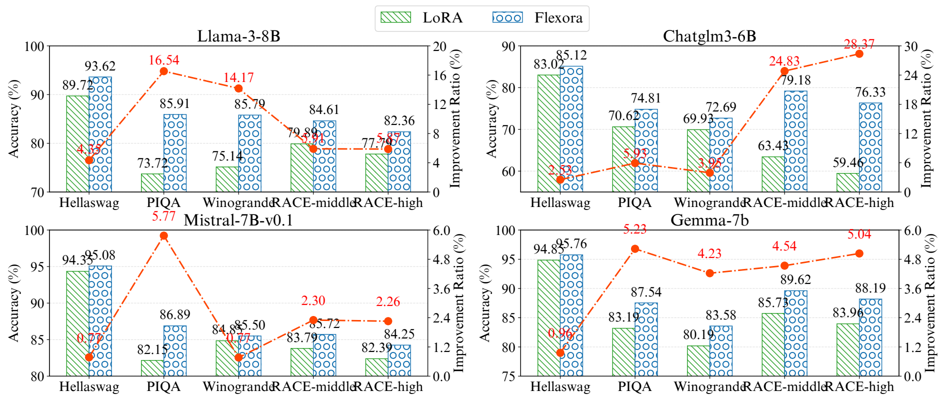

Figure 3: Comparison of the accuracy of various models (Llama-3-8B, ChatGLM3-6B, Mistral-7B-v0.1, and Gemma-7B) across different tasks. Bars with green diagonal stripes represent LoRA accuracy, while blue circles indicate Flexora accuracy. Notably, Flexora generally outperforms LoRA in most tasks and models, demonstrating its effectiveness in enhancing performance.

Table 3: Comparison of the accuracy of different randomly selected fine-tuning layers with the same number of fine-tuning layers. In this ablation study, we fixed the number of fine-tuning layers to match the number selected by Flexora, ensuring that the number of fine-tuning parameters remained constant while the layers were randomly selected for fine-tuning.

Methods	**Hellaswag**	**PIQA**	**Winogrande**	**RACE-mid**	**RACE-high**	**Average**
Random 1	92.97	82.91	80.98	83.98	81.10	84.39
Random 2	93.11	80.79	76.09	**85.45**	81.16	83.32
Random 3	92.52	80.47	83.50	84.54	81.93	84.59
Random (Avg.)	92.87	81.39	80.19	84.66	81.40	84.10
Flexora	**93.62**	**85.91**	**85.79**	84.61	**82.36**	**86.46**

outperform LoRA. The theoretical explanation for this result can be found in Sec. 6. Second, Flexora can automatically and flexibly select the specific layers for fine-tuning, targeting the most important layers to maximize performance and generalization. In Appendix C.5, we discuss the characteristics of the specific layers Flexora have chosen. The loss metrics are discussed in Appendix D.2.

Flexible Layer Selection in Flexora. In the second ablation experiment, we manually determine the number of fine-tuning layers and compare Flexora with random selection, highlighting the flexibility of Flexora. The results in Table 4 show that it can achieve the best performance regardless of the number of fine-tuning layers. The specific layers selected are shown in Table 12. The loss metrics are discussed in Appendix D.3. A noteworthy observation is that Flexora usually chooses the initial and final layers. An intuitive explanation is that the initial and final layers of the model have a significant impact on the data. The initial layers directly contact the original input, while the final layers are related to the model output, rendering them crucial. In addition, for the same downstream task, the input of the initial layer is consistent and closely coupled to the task, and the output of the final layer is also consistent. Focusing on optimizing these layers can improve learning efficiency. This conclusion has also been confirmed by other studies. LoRAShear[15] observed that the knowledge distribution in LLM is mainly concentrated in the initial and final layers. LASER[38] demonstrated that the loss gradients of the initial and final layers are steep, which is beneficial to the model during training. LISA[39] found that the weight norms of the initial and final layers are hundreds of times higher than those of the intermediate layers, indicating their increased importance.

6 Theoretical Insights

In this section, we provide theoretical explanations for *why Flexora (using only a subset of LoRA layers) can achieve excellent results*. We first introduce Theorem 1 below, and then simplify LoRA layers as linear layers in multi-layer perceptron (MLP) to derive our Proposition 2, aiming to offer insights for this question.

Table 4: Comparison of the accuracy of fine-tuning a subset of layers. In this ablation study, we standardized the number of layers to be fine-tuned and compared the performance of layers selected by Flexora against those selected randomly.

Methods	Hellaswag	PIQA	Winogrande	RACE-mid	RACE-high	Average
Random (6 Layers)	59.79	70.25	46.32	54.54	53.45	56.87
Flexora (First 6 Layers)	**60.04** (+0.25)	**77.20** (+6.95)	**57.54** (+11.22)	**69.71** (+15.17)	**58.35** (+4.90)	**64.57** (+7.70)
Random (12 Layers)	81.90	77.82	57.35	78.41	72.16	73.53
Flexora (First 12 Layers)	**88.85** (+6.95)	**79.71** (+1.89)	**65.82** (+8.47)	**79.42** (+1.01)	**72.33** (+0.17)	**77.23** (+3.70)
Random (18 Layers)	91.15	81.54	83.58	83.77	81.22	84.25
Flexora (First 18 Layers)	**91.31** (+0.16)	**82.21** (+0.67)	**84.69** (+1.11)	**84.07** (+0.30)	**81.53** (+0.31)	**84.76** (+0.51)
Random (24 Layers)	90.58	80.90	82.16	82.19	79.22	83.01
Flexora (First 24 Layers)	**91.01** (+0.43)	**81.21** (+0.31)	**82.87** (+0.71)	**83.53** (+1.34)	**80.22** (+1.00)	**83.77** (+0.76)

Theorem 1 (Theorem 3.8 in [40])**.** *Assume that $f(\cdot; z) \in [0, 1]$ is an L-Lipschitz and β-smooth loss function for every sample z. Suppose that we run stochastic gradient method (e.g., SGD) for T steps with monotonically non-increasing step sizes $\eta_t \le c/t$ ($t \in [T]$). In particular, omitting constant factors that depend on β, c, and L, we have $\mathcal{R}^{\text{test}}(\theta, \eta) \le \mathcal{R}^{\text{train}}(\theta, \eta) + \frac{T^{1-1/(\beta c+1)}}{m}$.*

Theorem 1 reveals that if all the conditions except for β in Theorem 1 remain the same, a smaller smoothness β will typically result in a smaller test error $\mathcal{R}^{\text{test}}(\theta, \eta)$, indicating a better generalization performance in practice. The specific definition of smoothness β can be found in Appendix A.2. To show how the number of LoRA layers is related to this β, we then follow the practice in [41] to prove our Proposition 2 below.

Proposition 2. *For an N-layer linear multi-layer perceptron (MLP): $y^{(N)} \triangleq \prod_{j=1}^{N} W^{(j)}\boldsymbol{x}$ with MSE function $\ell \triangleq (y^{(N)} - y)^2/2$ where y denotes the true label, let $\lambda^{(i)} = \left\|W^{(i)}\right\|$ for any $i \in [N]$, we then have $\left\|\frac{\partial \ell}{\partial W_1^{(i)}} - \frac{\partial \ell}{\partial W_2^{(i)}}\right\| \le \left(\prod_{j=1, j\neq i}^{N} \lambda^{(j)}\right)^2 \|\boldsymbol{x}\|^2 \left\|W_1^{(i)} - W_2^{(i)}\right\|$.*

The proof of Proposition 2 is in Appendix A.3. Given Proposition 2, the block-wise smoothness $\beta_i^{(N)}$ on layer $i \in [N]$ of an N-th layer MLP can be bounded by: $\beta_i^{(N)} \le \left(\prod_{j=1, j\neq i}^{N} \lambda^{(j)}\right)^2 \|\boldsymbol{x}\|^2$. From this bound, we can see that as the number of layers N increases, the upper bound of $\beta_i^{(N)}$ will also be increasing as $\lambda^{(i)} > 1$ for $i \in [N]$. Thus, shallow MLP of fewer layers are more likely to have smaller overall smoothness β. Thanks to this smaller overall smoothness β, shallow MLP of fewer layers are more likely to achieve a smaller generalization gap (i.e., the second term on the right-hand side of Theorem 1) than deep MLP with more layers. When the training error $\mathcal{R}^{\text{train}}(\theta, \eta)$ is the same, that is, both shallow and deep MLPs are fully trained to converge, the shallower MLP may have a lower test error $\mathcal{R}^{\text{test}}(\theta, \eta)$ and thus may exhibit better performance on downstream tasks.

Now, we can answer the question posed at the beginning of this section. When Flexora is fine-tuned with a subset of LoRA layers, it theoretically transforms a network with a deeper architecture into one with a shallower architecture. When sufficiently trained to convergence, the aforementioned theory suggests that a network with a shallower architecture can exhibit better generalization and performance on downstream tasks. In summary, the reason Flexora achieves excellent results is that it makes the model more suitable for downstream tasks.

7 Conclusion and Future Work

We introduce Flexora, a method to enhance the efficiency and effectiveness of fine-tuning large language models (LLMs) by automatically selecting the most critical layers, addressing overfitting, and improving performance. By modeling layer selection as an HPO problem and using UD to solve it, Flexora reduces computational overhead by updating only selected layers. Extensive experiments show that Flexora decreases parameters, mitigates overfitting, and outperforms baselines, with a theoretical explanation provided. Future work will explore the relationship between individual LLM layers and their impact on downstream tasks, and investigate optimal fine-tuning schemes for Flexora, particularly its integration with various LoRA-enhanced algorithms as detailed in Appendix C.2.

References

[1] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models, 2023.

[2] Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment, 2023.

[3] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models, 2022.

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

[5] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation, 2021.

[6] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021.

[7] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[8] Yichao Wu, Yafei Xiang, Shuning Huo, Yulu Gong, and Penghao Liang. Lora-sp: Streamlined partial parameter adaptation for resource-efficient fine-tuning of large language models, 2024.

[9] Zeyu Liu, Souvik Kundu, Anni Li, Junrui Wan, Lianghao Jiang, and Peter Anthony Beerel. Aflora: Adaptive freezing of low rank adaptation in parameter efficient fine-tuning of large models, 2024.

[10] Mingyang Zhang, Hao Chen, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, and Bohan Zhuang. Loraprune: Pruning meets low-rank parameter-efficient fine-tuning, 2024.

[11] Yang Lin, Xinyu Ma, Xu Chu, Yujie Jin, Zhibang Yang, Yasha Wang, and Hong Mei. Lora dropout as a sparsity regularizer for overfitting control, 2024.

[12] Yuzhu Mao, Siqi Ping, Zihao Zhao, Yang Liu, and Wenbo Ding. Enhancing parameter efficiency and generalization in large-scale models: A regularized and masked low-rank adaptation approach, 2024.

[13] Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning, 2023.

[14] Hongyun Zhou, Xiangyu Lu, Wang Xu, Conghui Zhu, Tiejun Zhao, and Muyun Yang. Lora-drop: Efficient lora parameter pruning based on output evaluation, 2024.

[15] Tianyi Chen, Tianyu Ding, Badal Yadav, Ilya Zharkov, and Luming Liang. Lorashear: Efficient large language model structured pruning and knowledge recovery, 2023.

[16] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search, 2019.

[17] Lan-Zhe Guo, Zhen-Yu Zhang, Yuan Jiang, Yu-Feng Li, and Zhi-Hua Zhou. Safe deep semi-supervised learning for unseen-class unlabeled data. In *International conference on machine learning*, pages 3897–3906. PMLR, 2020.

[18] Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science, 2016.

[19] Meta. Introducing meta llama 3: The most capable openly available LLM to date. *Meta Blog*, 2024.

[20] Yunqi Zhu, Xuebing Yang, Yuanyuan Wu, and Wensheng Zhang. Parameter-efficient fine-tuning with layer pruning on free-text sequence-to-sequence modeling, 2023.

[21] Fan Bao, Guoqiang Wu, Chongxuan Li, Jun Zhu, and Bo Zhang. Stability and generalization of bilevel programming in hyperparameter optimization, 2021.

[22] Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization, 2017.

[23] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimilano Pontil. Bilevel programming for hyperparameter optimization and meta-learning, 2018.

[24] Jie Fu, Hongyin Luo, Jiashi Feng, Kian Hsiang Low, and Tat-Seng Chua. Drmad: Distilling reverse-mode automatic differentiation for optimizing hyperparameters of deep neural networks, 2016.

[25] Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Gradient-based hyperparameter optimization through reversible learning, 2015.

[26] Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated back-propagation for bilevel optimization, 2019.

[27] Suvrit Sra, Sebastian Nowozin, and Stephen J Wright. *Optimization for machine learning*, page 351–368. Mit Press, 2011.

[28] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[29] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale, 2019.

[30] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations, 2017.

[31] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019.

[32] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019.

[33] Team GLM, :, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Juanzi Li, Lei Zhao, Lindong Wu, Lucen Zhong, Mingdao Liu, Minlie Huang, Peng Zhang, Qinkai Zheng, Rui Lu, Shuaiqi Duan, Shudan Zhang, Shulin Cao, Shuxun Yang, Weng Lam Tam, Wenyi Zhao, Xiao Liu, Xiao Xia, Xiaohan Zhang, Xiaotao Gu, Xin Lv, Xinghan Liu, Xinyi Liu, Xinyue Yang, Xixuan Song, Xunkai Zhang, Yifan An, Yifan Xu, Yilin Niu, Yuantao Yang, Yueyan Li, Yushi Bai, Yuxiao Dong, Zehan Qi, Zhaoyu Wang, Zhen Yang, Zhengxiao Du, Zhenyu Hou, and Zihan Wang. Chatglm: A family of large language models from glm-130b to glm-4 all tools, 2024.

[34] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.

[35] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Christian Muraru, Grigory Rozhdestvenskiy, Henryk

Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, Justin Mao-Jones, Katherine Lee, Kathy Yu, Katie Millican, Lars Lowe Sjoesund, Lisa Lee, Lucas Dixon, Machel Reid, Maciej Mikuła, Mateo Wirth, Michael Sharman, Nikolai Chinaev, Nithum Thain, Olivier Bachem, Oscar Chang, Oscar Wahltinez, Paige Bailey, Paul Michel, Petko Yotov, Rahma Chaabouni, Ramona Comanescu, Reena Jana, Rohan Anil, Ross McIlroy, Ruibo Liu, Ryan Mullins, Samuel L Smith, Sebastian Borgeaud, Sertan Girgin, Sholto Douglas, Shree Pandya, Siamak Shakeri, Soham De, Ted Klimenko, Tom Hennigan, Vlad Feinberg, Wojciech Stokowiec, Yu hui Chen, Zafarali Ahmed, Zhitao Gong, Tris Warkentin, Ludovic Peran, Minh Giang, Clément Farabet, Oriol Vinyals, Jeff Dean, Koray Kavukcuoglu, Demis Hassabis, Zoubin Ghahramani, Douglas Eck, Joelle Barral, Fernando Pereira, Eli Collins, Armand Joulin, Noah Fiedel, Evan Senter, Alek Andreev, and Kathleen Kenealy. Gemma: Open models based on gemini research and technology, 2024.

[36] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand, 2024. Association for Computational Linguistics.

[37] OpenCompass Contributors. Opencompass: A universal evaluation platform for foundation models. `https://github.com/open-compass/opencompass`, 2023.

[38] Pratyusha Sharma, Jordan T. Ash, and Dipendra Misra. The truth is in there: Improving reasoning in language models with layer-selective rank reduction, 2023.

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

[40] Moritz Hardt, Benjamin Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent, 2016.

[41] Yao Shu, Wei Wang, and Shaofeng Cai. Understanding architectures learnt by cell-based neural architecture search, 2020.

[42] Charles Blair. Problem complexity and method efficiency in optimization (as nemirovsky and db yudin). *Siam Review*, 27(2):264, 1985.

A Theorems and proofs

We first prove Proposition 1, then introduce the theorems proposed by [42] and [40], which reveal the properties of β-smooth, a necessary theoretical basis for proving Proposition 2. Finally, we prove Proposition 2.

A.1 Proof of proposition 1

The proof of Proposition 1 is expressed as follows:

Proof. It is easy to verify that

$$\frac{\partial \widehat{\alpha}^{(j)}}{\partial \alpha^{(i)}} = \begin{cases} \widehat{\alpha}^{(j)}(1 - \frac{1}{n}\widehat{\alpha}^{(j)}), & \text{if } j = i \\ -\frac{1}{n}\widehat{\alpha}^{(j)}\widehat{\alpha}^{(i)}, & \text{if } j \neq i \end{cases} .$$

Therefore, given that $\sum_{i=1}^{n} \widehat{\alpha}^{(i)} = n$

$$\begin{aligned} \sum_{i=1}^{n} \frac{\partial \widehat{\mathcal{R}}^{\text{val}}}{\partial \widehat{\alpha}^{(j)}} \frac{\partial \widehat{\alpha}^{(j)}}{\partial \alpha^{(i)}} &= \frac{\partial \widehat{\mathcal{R}}^{\text{val}}}{\partial \widehat{\alpha}^{(j)}} \left(\widehat{\alpha}^{(j)} - \frac{1}{n}\left(\widehat{\alpha}^{(j)}\right)^2 - \frac{1}{n} \sum_{i=1, i\neq j}^{n} \widehat{\alpha}^{(j)}\widehat{\alpha}^{(i)} \right) \\ &= \frac{\partial \widehat{\mathcal{R}}^{\text{val}}}{\partial \widehat{\alpha}^{(j)}} \left(\widehat{\alpha}^{(j)} - \frac{\widehat{\alpha}^{(j)}}{n} \sum_{i=1}^{n} \widehat{\alpha}^{(i)} \right) \\ &= 0 . \end{aligned}$$

When applying SGD to update α, we have

$$\sum_{i=1}^{n} \alpha^{(i)} - \eta \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial \widehat{\mathcal{R}}^{\text{val}}}{\partial \widehat{\alpha}^{(j)}} \frac{\partial \widehat{\alpha}^{(j)}}{\partial \alpha^{(i)}} = \sum_{i=1}^{n} \alpha^{(i)} .$$

That is, the updated α shares the same summation as the one before the updates, which therefore concludes our proof. □

A.2 Definition of β-Smooth

Definition 1. *β-smooth refers to the Lipschitz continuity of the gradient of the loss function, that is, for all w and w':*

$$\|\nabla f(w; z) - \nabla f(w'; z)\| \leq \beta \|w - w'\|$$

where $\|\cdot\|$ denotes the norm of the vector, and $f(w; z)$ is the loss function with parameter w for sample z.

Let $f_{\text{deep}}(w)$ and $f_{\text{shallow}}(w)$ be the loss functions for deep and shallow architectures, respectively. According to Definition 1, the relationship between β_{deep} and β_{shallow} illustrates the relationship between the generalization and performance of deep and shallow networks.

A.3 Proof of proposition 2

Abstract LLM into a layered network:[41] As shown in Figure 4, we abstract LLM into a hierarchical network, and the weight of each layer is represented by $W^{(i)}$. Figure 4 represents the general case. The output of the i-th layer network is:

$$\boldsymbol{y} = \prod_{j=1}^{n} W^{(j)} \boldsymbol{x}. \tag{5}$$

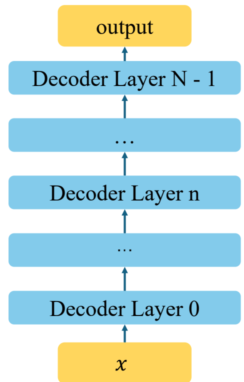

Figure 4: We present LLM as a hierarchical network. In this context, all parameters of a Decoder layer are represented as a weight matrix W for subsequent analysis.

Gradient analysis: For the abstract network, represented in Equation 5. The gradient of the loss function ℓ with respect to the weight $W^{(i)}$ is:

$$\frac{\partial \ell}{\partial W^{(i)}} = \left(\prod_{j=i+1}^{n} W^{(j)} \right) \frac{\partial \ell}{\partial \boldsymbol{y}^{(i)}} \boldsymbol{x} \left(\prod_{j=1}^{i-1} W^{(j)} \right). \tag{6}$$

The proof of Proposition 2 is expressed as follows:

Proof. For the abstract network, we begin with Definition 1:

$$\begin{aligned} &\left\| \frac{\partial \ell}{\partial W_1^{(i)}} - \frac{\partial \ell}{\partial W_2^{(i)}} \right\| \\ &= \left\| \left(\prod_{j=i+1}^{n} W^{(j)} \right) \left(\frac{\partial \ell}{\partial \boldsymbol{y}_1^{(i)}} - \frac{\partial \ell}{\partial \boldsymbol{y}_2^{(i)}} \right) \boldsymbol{x} \left(\prod_{j=1}^{i-1} W^{(j)} \right) \right\|. \end{aligned} \tag{7}$$

Taking MSE Loss as an example, for one predictions $y^{(N)}$ and their corresponding true values y:

$$\ell \triangleq (\boldsymbol{y}^{(N)} - \boldsymbol{y})^2/2, \tag{8}$$

therefore:

$$\frac{\partial \ell}{\partial \boldsymbol{y}^{(i)}} = \left(\boldsymbol{y}^{(N)} - \boldsymbol{y} \right) \prod_{j=i+1}^{N} W^{(j)}. \tag{9}$$

We select the MSE loss function and calculat the i-th layer of N layers network, Substituting Equation 9 into Equation 7:

$$\begin{aligned} \left\| \frac{\partial \ell}{\partial W_1^{(i)}} - \frac{\partial \ell}{\partial W_2^{(i)}} \right\| &= \left\| \left(\prod_{j=i+1}^{N} W^{(j)} \right)^2 \left(\boldsymbol{y}_1^{(N)} - \boldsymbol{y} - \boldsymbol{y}_2^{(N)} + \boldsymbol{y} \right) \boldsymbol{x} \left(\prod_{j=1}^{i-1} W^{(j)} \right) \right\| \\ &\leq \left(\frac{1}{\lambda^{(i)}} \left(\prod_{j=1}^{N} \lambda^{(j)} \right) \right) \left(\prod_{j=i+1}^{N} \lambda^{(j)} \right) \left\| \prod_{j=1}^{i-1} W^{(j)} \left(W_1^{(i)} - W_2^{(i)} \right) \boldsymbol{x}^2 \right\| \\ &\leq \left(\prod_{j=1, j \neq i}^{N} \lambda^{(j)} \right) \left\| \boldsymbol{x}^2 \right\| \left(\prod_{j=i+1}^{N} \lambda^{(j)} \right) \left(\prod_{j=1}^{i-1} \lambda^{(j)} \right) \left\| \left(W_1^{(i)} - W_2^{(i)} \right) \right\| \\ &\leq \left(\prod_{j=1, j \neq i}^{N} \lambda^{(j)} \right)^2 \left\| \boldsymbol{x}^2 \right\| \left\| W_1^{(i)} - W_2^{(i)} \right\|, \end{aligned} \tag{10}$$

which therefore concludes our proof. □

B Experimental setting

In the main experiment, we compared Flexora with the baseline. The datasets and experimental parameters are as follows:

B.1 Dataset

We added new templates to the original dataset to ensure the model could complete the required tasks and output formats. It is important to note that the added templates did not alter the original dataset, and special processing was performed for different LLMs. The specific examples are as follows:

Dataset Format of Hellaswag

```
dataset: Hellaswag
    dataset format:
    {
    "instruction": "{Article}\n
    Question: {Question}\n
    A. {Option A}\n
    B. {Option B}\n
    C. {Option C}\n
    D. {Option D}\n
    You may choose from 'A', 'B', 'C', 'D'.\n Answer:",
    "output": "{Answer}"
    }
    example:
    {
    "instruction": "A man is sitting on a roof. He\n
    Question: Which ending makes the most sense?\n
    A. is using wrap to wrap a pair of skis.\n
    B. is ripping level tiles off.\n
    C. is holding a Rubik's cube.\n
    D. starts pulling up roofing on a roof.\n
    You may choose from 'A', 'B', 'C', 'D'.\n Answer:",
    "output": "D"
    }
```

Dataset Format of PIQA

```
dataset: PIQA
    dataset format:
    {
    "instruction": "There is a single choice question.
    Answer the question by replying A or B.'\n
    Question: {Question}\n
    A. {Option A}\n
    B. {Option B}\n
    Answer:",
    "output": "{Answer}"
    }
    example:
    {
    "instruction": "There is a single choice question.
    Answer the question by replying A or B.'\n
    Question: When boiling butter, when it's ready, you can\n
    A. Pour it onto a plate\n
    B. Pour it into a jar\n
    Answer:",
    "output": "B"
    }
```

Dataset Format of Winogrande

```
dataset: Winogrande
    dataset format:
    {
    "instruction": "There is a single choice question,
    you need to choose the correct option to fill in the blank.
    Answer the question by replying A or B.'\n
    Question: {Question}\n
    A. {Option A}\n
    B. {Option B}\n
    Answer:",
    "output": "{Answer}"
    }
    example:
    {
    "instruction": "There is a single choice question,
    you need to choose the correct option to fill in the blank.
    Answer the question by replying A or B.'\n
    Question: Sarah was a much better surgeon than Maria so _ always got the
    easier cases.\n
    A. Sarah\n
    B. Maria\n
    Answer:",
    "output": "B"
    }
```

Dataset Format of RACE

```
dataset: RACE
    dataset format:
    {
    "instruction": "{Article}
    {Question}\n
    [ {Option A}, {Option B}\, {Option C},  {Option D}]",
    "output": "{Answer}"
    }
    example:
    {
    "instruction": "I am a psychologist. I first met Timothy, a quiet,
    overweight eleven-year-old boy, when his mother brought him to me to discuss
    his declining grades. A few minutes with Timothy were enough to confirm that
    his self-esteem  and general happiness were falling right along with _ .
    I asked about Timothy's typical day. He awoke every morning at six thirty
    so he could reach his school by eight and arrived home around four thirty each
    afternoon. He then had a quick snack, followed by either a piano lesson
    or a lesson with his math tutor. He finished dinner at 7 pm, and then he sat
    down to do homework for two to three hours. Quickly doing the math in my
    head, I found that Timothy spent an average of thirteen hours a day
    at a writing desk.\n
    What if Timothy spent thirteen hours a day at a sewing machine instead of
    a desk? We would immediately be shocked, because that would be called
    children being horribly mistreated. Timothy was far from being mistreated,
    but the mountain of homework he faced daily resulted in a similar consequence
    --he was being robbed of his childhood. In fact, Timothy had no time
    to do anything he truly enjoyed, such as playing video games, watching
    movies, or playing board games with his friends.\n
    Play, however, is a crucial part of healthy child development.
    It affects children's creativity, their social skills, and even their brain
    development. The absence of play, physical exercise, and freefrom social
    interaction takes a serious toll on many children. It can also cause
    significant health problems like childhood obesity, sleep problems
    and depression.\nExperts in the field recommend the minutes children
    spend on their homework should be no more than ten times the number
    of their grade level./nWhat did the writer think of Timothy after
    learning about his typical day?/n
    ['Timothy was very hardworking.',
    'Timothy was being mistreated.',
    'Timothy had a heavy burden.',
    'Timothy was enjoying his childhood.']",
    "output": "C"
    }
```

Table 5: Detailed experimental parameters. This table lists the specific parameters we used in the experiments for various methods. These parameters include the target module of LoRA (Lora Target), the maximum sequence length (Max Length), the number of samples for supervised fine-tuning (SFT Samples), the learning rate (LR), the number of search samples (Search Samples), the initial rank (Init Rank), the target rank (Target Rank), and the ratio of pruning (Ratio). All other parameters not listed here remain consistent across all experiments.

Methods	LoRA Target	Max Length	SFT Samples	LR	Search Samples	Init Rank	Target Rank	Ratio
LoRA	q & v Proj	1024	20000	0.0001	-	-	-	-
Flexora	q & v Proj	1024	20000	0.0001	20000	-	-	-
AdaLoRA	q & v Proj	1024	20000	0.0001	-	4	8	-
LoRA-drop	q & v Proj	1024	20000	0.0001	20000	-	-	-
LoRAShear	q & v Proj	1024	20000	0.0001	20000	-	-	0.5
Dora	q & v Proj	1024	20000	0.0001	20000	-	-	-
rsLoRA	q & v Proj	1024	20000	0.0001	20000	-	-	-
LoRAPrune	q & v Proj	1024	20000	0.0001	20000	-	-	0.5

B.2 Specific experimental parameters

Based on the Llama3-8B model configuration, several adjustments were made to optimize model performance. In the baseline model experiment, generation parameters were adjusted to ensure the correct output. In the LoRA experiment, adjustments to the generation parameters were retained, and LoRA-related parameters were adjusted. In the Flexora experiment, the size of the validation set was adjusted to control the time required to search for the optimal layer. In the AdaLoRA experiment, the initial rank size was modified to ensure that the fine-tuning parameters are consistent with Flexora. In the LoRA-drop experiment, the number of fine-tuning layers was set to be consistent with Flexora to ensure that the fine-tuning parameters are consistent. In the LoRAShear experiment, the pruning ratio was modified, where the parameter amount with a pruning ratio of 50% is consistent with Flexora. For specific experimental parameters, see the table 5.

B.3 Other LLMs experimental parameters

In order to explore the versatility and scalability of Flexora, we conducted experiments on multiple different LLMs. The specific training parameters are shown in Table 6.

C More results

C.1 The results of other LLMs experiment

Wide Applicability of Flexora. According to the parameter settings in Table 6, the verification results for various LLMs are presented in Table 7. The selected LLMs include Llama3-8B, Llama-7B, Llama2-7B, ChatGLM3-6B, Mistral-7B-v0.1, Gemma-7B, Zephyr-7B-beta, Vicuna-7B-v1.5, XuanYuan-6B, Qwen1.5-7B, and Yi-6B. These models demonstrate unique characteristics in terms of training data, architecture design, and optimized training. First, the models utilize varied training data, leading to differences in data distribution. Additionally, some models have enhanced attention mechanisms: Mistral-7B-v0.1 employs grouped query attention (GQA) and sliding window attention (SWA), while ChatGLM3-6B features a special attention design to support tool calling and code execution capabilities. Activation functions vary across these models. Llama3-8B uses the SwiGLU activation function, inspired by the PaLM model, to improve performance and convergence speed, while ChatGLM3-6B uses the Swish activation function. Furthermore, differences in reasoning optimization and multilingual capabilities contribute to varied reasoning abilities across fields. The experimental result of each model is shown in Table 7, which presents the scores of each model on different downstream tasks after LoRA and Flexora fine-tuning. It should be noted that all models fine-tuned using LoRA will have a certain degree of overfitting, while Flexora can effectively identify and analyze unnecessary layers in specific downstream tasks and prune them to reduce model overfitting. After optimization by Flexora, these LLMs showed significant performance

Table 6: Detailed LLM experiment parameters. This table provides a comprehensive overview of the specific parameters used for different large language models (LLMs) in our experiments. These parameters include the LoRA alpha value (LoRA Alpha), the dropout rate of LoRA (LoRA Dropout), the rank used in LoRA (LoRA Rank), and the target module of LoRA (LoRA Target). In addition, the table lists the specific templates used for each LLM, which are derived from Llama-factory (Template). For experiments involving different downstream tasks using the same model, all other parameters are kept consistent to ensure fair comparison and best performance.

LLM	LoRA Alpha	LoRA Dropout	LoRA Rank	LoRA Target	Tamplate (From Llama-factory)
Llama3	16	0	8	q & v Proj	llama3
Llama	16	0	8	q & v Proj	defult
Llama2	16	0	8	q & v Proj	llama2
chatglm3	16	0	8	query_key_value	chatglm3
Mistral-v0.1	16	0	8	q & v Proj	mistral
gemma	16	0	8	q & v Proj	gemma
zephyr	16	0	8	q & v Proj	zephyr
vicuna	16	0	8	q & v Proj	vicuna
xuanyuan	16	0	8	q & v Proj	xuanyuan
qwen1.5	16	0	8	q & v Proj	qwen
yi	16	0	8	q & v Proj	yi

improvements on downstream tasks. In particular, models that originally performed poorly on some tasks, such as ChatGLM3-6B, experienced significant improvements, achieving more than a 15% increase on the RACE-mid and RACE-high tasks. This improvement is attributable to the key layer selection by Flexora and efficient model learning. In summary, Flexora is applicable across Transformer models of various structures, excels in diverse tasks, and effectively enhances areas where model capabilities are lacking.

C.2 The results of other LoRAs experiment

Strong Scalability of Flexora. Recently, as emphasized in the Introduction 1, numerous LoRA improvement methods have been proposed, achieving excellent performance in specific fine-tuning tasks. In this section, the potential for combining Flexora with other emerging LoRA algorithms is explored. Two promising LoRA variants were selected from different approaches, each demonstrating impressive performance. Specifically, DoRA (Weight-Decomposed Low-Rank Adaptation) achieves low-rank adaptation through weight decomposition, and rsLoRA (Rank-Stabilized LoRA) addresses the slow training speed of traditional LoRA by introducing a rank-stabilized scaling factor when increasing rank. These methods primarily address the parameter overfitting problem within LoRA parameters but overlook the overall overfitting issue. These methods were innovatively combined with Flexora to first address the overall overfitting problem and then tackle the overfitting of the remaining LoRA parameters, resulting in notable performance improvements. The specific experimental results are shown in Table 8. The results indicate that Flexora integrates well with both DoRA and rsLoRA, effectively mitigating the overfitting problem of LLMs and improving performance with training on less than half of the parameters. The specific implementation entails replacing LoRA with DoRA or rsLoRA for inner layer optimization during the flexible layer selection stage, with the outer layer optimization remaining unchanged. These adjustments are achievable through straightforward modifications. The results demonstrate that Flexora exhibits strong scalability when combined with algorithms that enhance LoRA parameters, highlighting its significant potential.

C.3 Comparison with LoRAShear

Better Performance of Flexora. In this section, the accuracy of Flexora is compared with that of LoRAShear across various datasets, with specific results presented in Table 9. Since LoRAShear is not open source and poses challenges for direct experimentation, the comparison relies on the experimental configurations and results reported in the LoRAShear paper. Notably, Flexora can freely adjust the selected layers according to the dataset, achieving an average pruning parameter

Table 7: Detailed comparison of the accuracy of different LLMs. This table presents a comprehensive comparison of the accuracy results obtained by fine-tuning various mainstream Large Language Models (LLMs) using Flexora and LoRA methods. The accuracy metrics are reported across multiple benchmark datasets, including HellaSwag, PIQA, Winogrande, RACE-mid, and RACE-high. The average accuracy across all datasets is also provided. The exact values of accuracy improvements for each method, highlighted in red, indicate the performance gains achieved.

Methods	**Hellaswag**	**PIQA**	**Winogrande**	**RACE-mid**	**RACE-high**	**Average**
Llama3-8B-LoRA	89.72	73.72	75.14	79.89	77.79	79.25
Llama3-8B-Flexora	93.62	85.91	85.79	84.61	82.36	86.46
	(+3.90)	(+12.19)	(+10.65)	(+4.72)	(+4.57)	(+7.21)
Llama-7B-LoRA	76.10	69.80	67.01	75.69	70.81	71.88
Llama-7B-Flexora	85.28	71.93	74.11	81.62	78.62	78.31
	(+9.18)	(+2.13)	(+7.10)	(+5.93)	(+7.81)	(+6.43)
Llama2-7B-LoRA	79.60	75.90	78.60	79.32	75.07	77.70
Llama2-7B-Flexora	90.89	81.72	82.85	84.89	83.19	84.71
	(+11.29)	(+5.82)	(+4.25)	(+5.57)	(+8.12)	(+7.01)
Chatglm3-6B-LoRA	83.02	70.62	69.93	63.43	59.46	69.29
Chatglm3-6B-Flexora	85.12	74.81	72.69	79.18	76.33	77.63
	(+2.10)	(+4.19)	(+2.76)	(+15.75)	(+16.87)	(+8.33)
Mistral-7B-v0.1-LoRA	94.35	82.15	84.85	83.79	82.39	85.51
Mistral-7B-v0.1-Flexora	95.08	86.89	85.50	85.72	84.25	87.49
	(+0.73)	(+4.74)	(+0.65)	(+1.93)	(+1.86)	(+1.98)
Gemma-7B-LoRA	94.85	83.19	80.19	85.73	83.96	85.58
Gemma-7B-Flexora	95.76	87.54	83.58	89.62	88.19	88.94
	(+0.91)	(+4.35)	(+3.39)	(+3.89)	(+4.23)	(+3.35)
Zephyr-7B-beta-LoRA	93.77	75.03	78.37	83.45	82.25	82.57
Zephyr-7B-beta-Flexora	95.05	85.58	84.95	86.19	84.30	87.21
	(+1.28)	(+10.55)	(+6.58)	(+2.74)	(+2.05)	(+4.64)
Vicuna-7B-v1.5-LoRA	87.64	69.48	63.85	67.30	73.90	72.43
Vicuna-7B-v1.5-Flexora	90.43	79.49	76.06	82.94	81.90	82.16
	(+2.79)	(+10.01)	(+12.21)	(+15.64)	(+8.00)	(+9.73)
XuanYuan-6B-LoRA	82.38	74.16	65.27	78.04	72.11	74.39
XuanYuan-6B-Flexora	88.41	79.43	73.40	84.89	80.70	81.37
	(+6.03)	(+5.27)	(+8.13)	(+6.85)	(+8.59)	(+6.97)
Qwen1.5-7B-LoRA	91.75	75.03	78.14	87.59	81.36	82.77
Qwen1.5-7B-Flexora	91.96	84.33	80.69	89.90	87.08	86.79
	(+0.21)	(+9.30)	(+2.55)	(+2.31)	(+5.72)	(+4.02)
Yi-6B-LoRA	89.46	78.29	76.01	80.02	85.13	81.78
Yi-6B-Flexora	92.24	84.82	84.96	88.72	86.91	87.53
	(+2.78)	(+6.53)	(+8.95)	(+8.70)	(+1.78)	(+5.75)

Table 8: Detailed comparison of the accuracy of the combination of Flexora and different LoRA algorithms on Llama3-8B. This table presents a detailed comparison of the accuracy results obtained by integrating Flexora with various improved LoRA algorithms, including DoRA and rsLoRA, while maintaining other experimental settings constant. The accuracy metrics are reported across multiple benchmark datasets, including HellaSwag, PIQA, Winogrande, RACE-mid, and RACE-high, with the average accuracy across all datasets also provided. The results are compared against those obtained from direct fine-tuning without Flexora. The experimental findings indicate that the application of Flexora can significantly reduce model overfitting and enhance overall performance.

Methods	Hellaswag	PIQA	Winogrande	RACE-mid	RACE-high	Average
LoRA	89.72	76.39	82.24	85.86	80.99	83.04
Flexora (w/ LoRA)	**93.62**	**85.91**	**85.79**	**84.61**	**82.36**	**86.46**
rsLoRA	94.33	87.21	85.32	87.60	84.36	87.76
Flexora (w/ rsLoRA)	**94.83**	**87.58**	**86.69**	**88.21**	**85.46**	**88.55**
DoRA	93.62	85.75	84.77	86.77	83.39	86.86
Flexora (w/ DoRA)	**94.10**	**86.05**	**86.32**	**87.12**	**84.45**	**87.61**

Table 9: Detailed comparison of commonsense reasoning task accuracy. This table provides a comprehensive comparison of the accuracy results for various methods applied to common sense reasoning tasks, conducted on the Llama-7B model. The methods compared include the pre-trained model, LoRA, LoRAShear with different pruning ratios (0.5), and Flexora. The accuracy metrics are reported across multiple benchmark datasets, including BoolQ, PIQA, HellaSwag, Winogrande, ARC-e, ARC-c, and OBQA. The average accuracy across all datasets is also provided. The “Ratio" column represents the ratio of parameter pruning in LoRAShear.

Methods	BoolQ	PIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Average
Pre-trained	57.98	60.94	34.35	52.25	31.82	27.30	35.80	42.92
LoRA	67.76	69.80	76.10	67.01	67.21	35.23	38.60	60.24
LoRAShear (Ratio = 0.5)	63.40	72.15	49.83	56.40	49.45	34.31	35.86	51.63
Flexora	**73.54**	**71.93**	**85.28**	**74.11**	**71.22**	**45.64**	**39.86**	**65.94**

rate of 50%. Consequently, under the same pruning rate, Flexora outperforms by 14% (Ratio = 0.5). Experiments have shown that under the same pruning rate, Flexora can achieve better performance.

C.4 Different search sample

Flexibility of Flexora in search sample . In Flexora, search time is managed by adjusting the maximum number of search samples (corresponding to the size of the validation dataset) to align with the requirements of the downstream task. In Table 10, we explore the relationship between different numbers of search samples, downstream task performance, and search time. For simpler datasets like Hellaswag and PIQA, a 10-minute search with 1,000 samples significantly improves performance. For more challenging tasks, at least 1 hour of search time is required for 5,000 samples. In more difficult tasks, using too few samples can prevent validation loss from converging. To optimize performance, it is recommended to dynamically adjust the number of search samples based on the convergence of the validation loss. In summary, for simpler downstream tasks, Flexora can be rapidly applied to reduce model overfitting significantly and enhance performance. For more challenging downstream tasks, Flexora balances performance and training resources by adjusting the number of search samples.

C.5 Selection of layers

For different LLMs and datasets, the layers chosen by Flexora vary due to the different parameters learned in the pre-training stage and the diversity of downstream tasks. In Table 11, Table 12, Table 13, Table 14, and Table 15, we show the layers chosen by Flexora in all experiments and the corresponding training parameters. In this section, the preferences of the layers chosen by Flexora are analyzed in detail, providing layer-wise insights for LLMs.

Table 10: Detailed analysis of the impact of different numbers of search samples on the Flexora accuracy of Llama3-8B. This table investigates how varying the number of search samples, i.e., different validation dataset sizes, affects the performance of Flexora. The accuracy metrics are reported across multiple benchmark datasets, including HellaSwag, PIQA, Winogrande, RACE-mid, and RACE-high, with the average accuracy across all datasets also provided. The number of search samples tested includes 1000, 2000, 5000, 10000, and 200000. All experimental conditions remain unchanged except for the size of the validation set, allowing for a focused analysis on the impact of search sample size on model performance.

# Samples	Hellaswag	PIQA	Winogrande	RACE-mid	RACE-high	Average
1000	93.00	80.52	76.40	76.74	72.93	79.92
2000	92.29	81.98	74.11	80.15	78.82	81.47
5000	93.17	82.48	81.53	**84.82**	82.16	84.83
10000	93.47	84.07	83.11	84.17	81.56	85.28
20000	**93.62**	**85.91**	**85.79**	84.61	**82.36**	**86.46**

The Effectiveness of Flexora Comes from Reducing Overfitting. In Table 11, the layers and parameter amounts selected by different LoRA methods are presented. A comparison between LoRA-drop and Flexora reveals that Flexora is more effective. LoRA-drop tends to select the later layers, as these outputs exhibit a larger two-norm, aligning with Proposition 2. This result suggests that layers selected during fine-tuning should not concentrate in a specific range but rather be distributed across various ranges, fully utilizing the extensive knowledge system of LLMs. Comparing LoRA with DoRA and rsLoRA shows that LoRA selects more layers, requiring more training parameters but yielding worse performance. This suggests a higher degree of overfitting when Flexora is applied to LoRA compared to the other two methods. Therefore, using more advanced LoRA improvement algorithms can significantly reduce overfitting and enhance performance, underscoring the importance of the fine-tuning approach. Interestingly, certain layers are consistently fine-tuned in the same downstream task, regardless of whether LoRA, DoRA, or rsLoRA is used. For example, in Hellaswag, layers [0, 1, 2, 4, 14, 15, 19, 20, 21, 23, 26, 27, 28, 29, 31] are consistently selected, suggesting these layers are crucial for this task or represent general knowledge layers (see the next two paragraphs for details), closely related to the LLM itself .

General Knowledge Layers. In Table 12, the layers and parameters selected in the second ablation study are shown. Observing the "Select first 6 layers by Flexora" row reveals that certain layers, such as [27, 28], are crucial for any downstream task. These layers may store general knowledge, suggesting that their fine-tuning could enhance the performance across most downstream tasks.

Downstream task-specific layers. Table 13 displays the layers and parameter amounts selected by various LLMs for different downstream tasks. As evident from the table, the same model utilizes the aforementioned general knowledge layers across different tasks. Additionally, unique layers for each downstream task, termed downstream task-specific layers, are predominantly found in the first and last layers. The distinction between general knowledge layers and downstream task-specific layers can be attributed to the self-attention mechanism, which effectively differentiates these layers. In the self-attention mechanism, similar knowledge is aggregated, leading to this layer differentiation. Furthermore, concerning downstream task-specific layers, two conclusions are drawn: (a) Fewer layers are selected for simpler datasets to minimize overfitting. (b) Typically, the initial and final layers are selected for a given dataset. This selection pattern may stem from the initial layer processing the original input and the final layer generating the model's output representation. Given the consistent and predefined input and output, learning these parameters is deemed effective.

Poor Effects with No Critical Layers Tables 14 and 15 serve as evidence for the existence of downstream task-specific and general knowledge layers. Failure to select these layers, due to reasons like random selection or lack of convergence, leads to poor performance.

In summary, it is evident that almost all LLMs feature downstream task-specific layers and general knowledge layers. Fine-tuning these layers effectively mitigates model overfitting and enhances both generalization and performance. Fortunately, Flexora accurately and efficiently identifies both the downstream task-specific layers and the general knowledge layers.

D Loss

This section presents the training, evaluation, and validation loss during the Flexora flexible layer selection and fine-tuning stages, accompanied by intuitive explanations.

D.1 Effectiveness of Flexora.

Figure 5 plots the training and validation loss curves for Llama-8B during the flexible layer selection stage across four different datasets over one epoch. Both inner and outer layer optimizations are observed to converge well during the flexible layer selection stage, demonstrating the effectiveness of Flexora.

D.2 Flexora can Correctly Identify Critical Layers.

Figures 6, 7, 8, and 9 depict the training and evaluation loss from the first ablation study. In all experiments, the training loss converges effectively, demonstrating robust training performance. However, variations in evaluation loss underscore the model's generalization capabilities. Flexora generally surpasses methods that randomly select an equivalent number of layers, demonstrating its ability to accurately identify critical layers for more effective improvements.

D.3 Flexora can Reduce Overfitting.

Figures 10, 11, 12, and 13 present the training and evaluation loss from the sencond ablation study. Consistent with previous experiments, the training loss converges, indicating a strong training effect on the training set. Notably, the 24-layer (red) model consistently shows the lowest training loss, suggesting optimal learning, whereas the 6-layer (blue) model consistently records the highest, indicating poorer training performance. However, differences in evaluation loss reveal variations in model generalization across different layers. The 18-layer (green) model consistently exhibits the lowest evaluation loss, indicating superior generalization and downstream task performance, corroborated by actual results. The 24-layer (red) model's evaluation loss consistently exceeds that of the 18-layer (green) model, suggesting significant overfitting. Similarly, the 6-layer (blue) model consistently records the highest evaluation loss, indicative of underfitting.

In summary, too few training layers can lead to underfitting and poor performance, as seen in the 6-layer (blue) model. Conversely, too many layers can also result in overfitting, as evidenced by the 24-layer (red) model's performance. However following the selection strategy of Flexora, choosing the right number of layers can minimize overfitting and improve performance

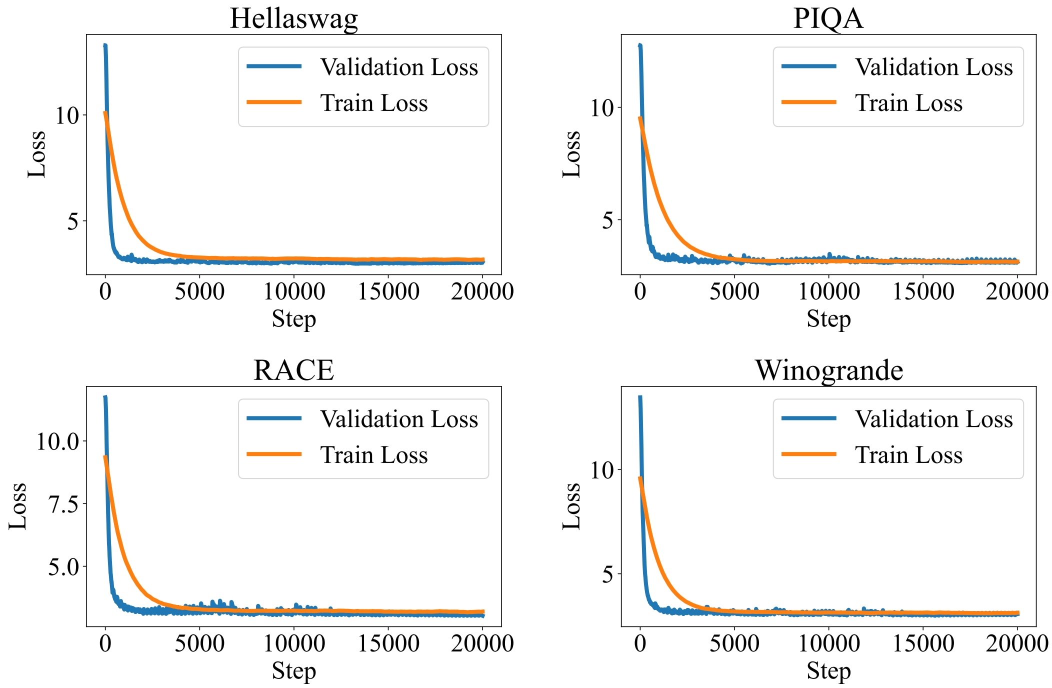

Figure 5: Training and validation loss during the flexible layer selection phase. The figure shows the training and validation loss over 20,000 steps for four different datasets (Hellaswag, PIQA, RACE, and Winogrande), where the batch size at each step is 1. The blue line shows the validation loss and the orange line shows the training loss. These plots visually compare how the performance of the models changes during the flexible layer selection phase, highlighting the convergence behavior.

Table 11: Comprehensive overview of layer selection strategies in main experiments. This table presents a detailed breakdown of the layer selection strategies used in different experiments involving the Llama3-8B model and its variants (Flexora, LoRA-drop, DoRA + Flexora, and rsLoRA + Flexora). For each model, the specific datasets utilized (HellaSwag, PIQA, RACE, and Winogrande) are listed along with the corresponding layers selected for each dataset. The "Layer selection" column provides the indices of the layers chosen for each experiment, indicating the specific layers of the model that were fine-tuned or modified. Additionally, the "Parameter(M)" column indicates the total number of parameters (in millions) used in each configuration. This detailed breakdown allows for a clear understanding of the experimental setup, the layer selection process, and the parameter allocation across different models and datasets, facilitating a deeper analysis of the impact of these strategies on model performance.

Methods	Dataset	Layer selection	Parameter(M)
Llama3-8B + Flexora	Hellaswag	[0, 1, 2, 3, 4, 5, 6, 14, 15, 19, 20, 21, 23, 24, 26, 27, 28, 29, 31]	2.0
	PIQA	[1, 2, 3, 4, 5, 7, 8, 9, 14, 20, 25, 26, 27, 28, 29, 30]	1.7
	RACE	[0, 1, 2, 3, 4, 7, 8, 9, 12, 14, 25, 26, 27, 28, 29, 31]	1.7
	Winogrande	[0, 1, 2, 3, 4, 16, 20, 22, 23, 24, 25, 26, 27, 28, 29, 31]	1.7
Llama3-8B + LoRA-drop	Hellaswag	[13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]	2.0
	PIQA	[16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]	1.7
	RACE	[16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]	1.7
	Winogrande	[16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]	1.7
Llama3-8B + DoRA + Flexora	Hellaswag	[0, 1, 2, 4, 5, 14, 15, 19, 20, 21, 23, 26, 27, 28, 29, 31]	1.8
	PIQA	[0, 1, 2, 4, 7, 23, 24, 25, 26, 27, 28, 29, 31]	1.5
	RACE	[1, 3, 4, 7, 9, 12, 14, 23, 25, 27, 28, 29, 31]	1.3
	Winogrande	[0, 1, 2, 3, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 31]	1.7
Llama3-8B + rsLoRA + Flexora	Hellaswag	[0, 1, 2, 4, 6, 14, 15, 19, 20, 21, 23, 25, 26, 27, 28, 29, 31]	1.8
	PIQA	[0, 1, 2, 3, 15, 20, 21, 25, 26, 27, 28, 29, 31]	1.3
	RACE	[0, 1, 2, 3, 7, 8, 12, 13, 25, 26, 27, 28, 29, 31]	1.5
	Winogrande	[1, 2, 3, 6, 14, 15, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 31]	1.9

Table 12: Detailed display of selected layers in the second ablation study. In the second ablation experiment, we manually determined the number of fine-tuning layers and contrasted the performance of Flexora with random layer selection strategies. This table presents the results of this experiment, showcasing different configurations where a specific number of layers (6, 12, 18, and 24) were selected for fine-tuning. For each configuration, the table compares the layers selected by Flexora with those selected randomly. The datasets used in this experiment include HellaSwag, PIQA, RACE, and Winogrande. The “Layer selection" column lists the indices of the layers chosen for fine-tuning in each dataset, while the “Parameter(M)" column indicates the total number of parameters (in millions) used in each configuration. This detailed breakdown provides insights into how different layer selection strategies, with a manually determined number of fine-tuning layers, impact the performance of model across different datasets, facilitating a comprehensive comparison between Flexora and random selection methods.

Methods	**Dataset**	**Layer selection**	**Parameter(M)**
Select first 6 layers by Flexora	Hellaswag	[0, 26, 27, 28, 29, 31]	0.6
	PIQA	[2, 4, 26, 27, 28, 29]	0.6
	RACE	[0, 7, 12, 27, 28, 29]	0.6
	Winogrande	[22, 23, 24, 26, 27, 28]	0.6
Random selection 6 layers	Hellaswag	[2, 4, 11, 19, 23, 25]	0.6
	PIQA	[2, 4, 11, 19, 23, 25]	0.6
	RACE	[2, 4, 11, 19, 23, 25]	0.6
	Winogrande	[2, 4, 11, 19, 23, 25]	0.6
Select first 12 layers by Flexora	Hellaswag	[0, 2, 3, 14, 15, 21, 23, 26, 27, 28, 29, 31]	1.3
	PIQA	[1, 2, 3, 4, 7, 20, 25, 26, 27, 28, 29, 30]	1.3
	RACE	[0, 1, 3, 7, 8, 12, 13, 25, 27, 28, 29, 31]	1.3
	Winogrande	[0, 3, 20, 22, 23, 24, 25, 26, 27, 28, 29, 31]	1.3
Random selection 12 layers	Hellaswag	[1, 3, 4, 12, 14, 18, 20, 21, 22, 27, 29, 31]	1.3
	PIQA	[1, 3, 4, 12, 14, 18, 20, 21, 22, 27, 29, 31]	1.3
	RACE	[1, 3, 4, 12, 14, 18, 20, 21, 22, 27, 29, 31]	1.3
	Winogrande	[1, 3, 4, 12, 14, 18, 20, 21, 22, 27, 29, 31]	1.3
Select first 18 layers by Flexora	Hellaswag	[0, 1, 2, 3, 4, 5, 6, 14, 15, 19, 21, 23, 26, 27, 28, 29, 30, 31]	1.9
	PIQA	[0, 1, 2, 3, 4, 5, 7, 8, 19, 20, 23, 25, 26, 27, 28, 29, 30, 31]	1.9
	RACE	[0, 1, 2, 3, 4, 7, 8, 9, 10, 12, 13, 15, 25, 27, 28, 29, 30, 31]	1.9
	Winogrande	[0, 1, 3, 5, 7, 9, 15, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 31]	1.9
Random selection 18 layers	Hellaswag	[1, 2, 5, 8, 9, 10, 12, 13, 17, 18, 20, 21, 22, 23, 24, 25, 26, 30]	1.9
	PIQA	[1, 2, 5, 8, 9, 10, 12, 13, 17, 18, 20, 21, 22, 23, 24, 25, 26, 30]	1.9
	RACE	[1, 2, 5, 8, 9, 10, 12, 13, 17, 18, 20, 21, 22, 23, 24, 25, 26, 30]	1.9
	Winogrande	[1, 2, 5, 8, 9, 10, 12, 13, 17, 18, 20, 21, 22, 23, 24, 25, 26, 30]	1.9
Select first 24 layers by Flexora	Hellaswag	[0, 1, 2, 3, 4, 5, 6, 11, 12, 13, 14, 15, 18, 19, 20, 21, 23, 24, 26, 27, 28, 29, 30, 31]	2.6
	PIQA	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 15, 18, 19, 20, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31]	2.6
	RACE	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 23, 24, 25, 27, 28, 29, 30, 31]	2.6
	Winogrande	[0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 15, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]	2.6
Random selection 24 layers	Hellaswag	[0, 1, 4, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 21, 23, 25, 26, 27, 28, 30, 31]	2.6
	PIQA	[0, 1, 4, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 21, 23, 25, 26, 27, 28, 30, 31]	2.6
	RACE	[0, 1, 4, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 21, 23, 25, 26, 27, 28, 30, 31]	2.6
	Winogrande	[0, 1, 4, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 21, 23, 25, 26, 27, 28, 30, 31]	2.6

Table 13: Comprehensive overview of layer selection strategies and parameter allocation in various experiments. This table provides an in-depth breakdown of the layer selection strategies employed across different models and datasets in the experiments. The models tested include Llama3-8B, Chatglm3-6B, Mistral-7B-v0.1 and others, all combined with Flexora. For each model, the specific datasets used (HellaSwag, PIQA, RACE, and Winogrande) are listed along with the corresponding layers selected for each dataset. The "Layer selection" column details the indices of the layers chosen for each experiment, indicating the specific layers of the model that were fine-tuned or modified. Additionally, the "Parameter(M)" column indicates the total number of parameters (in millions) used in each configuration. This detailed breakdown allows for a clear understanding of the experimental setup, the layer selection process, and the parameter allocation across different models and datasets, facilitating a deeper analysis of the impact of these strategies on model performance.

Methods	Dataset	Layer selection	Parameter(M)
Llama3-8B + Flexora	Hellaswag	[0, 1, 2, 3, 4, 5, 6, 14, 15, 19, 20, 21, 23, 24, 26, 27, 28, 29, 31]	2.0
	PIQA	[1, 2, 3, 4, 5, 7, 8, 9, 14, 20, 25, 26, 27, 28, 29, 30]	1.7
	RACE	[0, 1, 2, 3, 4, 7, 8, 9, 12, 14, 25, 26, 27, 28, 29, 31]	1.7
	Winogrande	[0, 1, 2, 3, 4, 16, 20, 22, 23, 24, 25, 26, 27, 28, 29, 31]	1.7
Chatglm3-6B + Flexora	Hellaswag	[1, 2, 3, 4, 5, 6, 7, 10, 12, 13, 16, 18, 20]	0.9
	PIQA	[0, 1, 2, 3, 5, 6, 7, 8, 9, 19, 21, 23, 25, 27]	1.0
	RACE	[2, 6, 8, 9, 10, 11, 14, 15, 16, 17, 18, 20, 23, 26]	1.0
	Winogrande	[0, 2, 6, 8, 9, 11, 12, 13, 16, 17, 18, 20, 25, 26]	1.0
Mistral-7B-v0.1 + Flexora	Hellaswag	[0, 1, 2, 3, 4, 5, 6, 7, 14, 22, 26, 27, 30]	1.5
	PIQA	[6, 8, 14, 17, 18, 22, 23, 24, 25, 26, 27, 28, 29, 30]	1.7
	RACE	[0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 17, 30, 31]	1.7
	Winogrande	[0, 1, 2, 3, 4, 5, 6, 7, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]	1.9
Gemma-7B + Flexora	Hellaswag	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 14, 15, 16, 18, 20, 23, 27]	1.9
	PIQA	[0, 1, 8, 9, 10, 12, 15, 16, 17, 20, 21, 22, 23, 24, 25, 26, 27]	1.9
	RACE	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 15, 16]	1.4
	Winogrande	[0, 1, 2, 3, 4, 5, 6, 7, 8, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27]	2.1
Vicuna-7B-v1.5 + Flexora	Hellaswag	[0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12]	1.6
	PIQA	[1, 2, 3, 5, 7, 8, 11, 12, 13, 14, 21, 31]	1.6
	RACE	[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]	1.6
	Winogrande	[0, 2, 3, 4, 6, 8, 9, 12, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]	2.6
Zephyr-7B-beta + Flexora	Hellaswag	[1, 13, 15, 17, 18, 22, 23, 24, 25, 26, 27, 28, 30, 31]	1.5
	PIQA	[2, 3, 6, 7, 14, 15, 16, 17, 22, 26, 27, 28]	1.4
	RACE	[1, 2, 4, 6, 7, 9, 11, 13, 14, 17, 26, 30, 31]	1.4
	Winogrande	[1, 3, 5, 6, 8, 13, 27, 28, 29, 30, 31]	1.2
Yi-6B + Flexora	Hellaswag	[0, 1, 2, 3, 4, 6, 8, 9, 10, 19, 20, 21, 22]	1.3
	PIQA	[1, 2, 3, 5, 6, 7, 8, 9, 12, 13, 15, 16, 17, 18, 20, 23]	1.6
	RACE	[1, 3, 5, 6, 7, 9, 11, 12, 13, 14, 17, 21]	1.2
	Winogrande	[0, 1, 2, 3, 5, 6, 7, 11, 23, 26, 27, 30, 31]	1.3
Llama-7B + Flexora	Hellaswag	[0, 1, 2, 4, 5, 6, 8, 12, 16, 30, 31]	1.4
	PIQA	[2, 12, 14, 15, 16, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]	2.1
	RACE	[4, 5, 6, 7, 8, 10, 11, 23, 30, 31]	1.3
	Winogrande	[0, 2, 3, 6, 7, 8, 10, 11, 13, 16, 23, 28, 29, 30, 31]	2.0
Llama2-7B + Flexora	Hellaswag	[0, 1, 2, 3, 4, 5, 6, 7, 8, 12]	1.3
	PIQA	[0, 1, 2, 3, 7, 8, 11, 13, 14, 21, 24, 29, 30, 31]	1.8
	RACE	[0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16]	1.8
	Winogrande	[0, 1, 3, 4, 8, 14, 15, 16, 17, 20, 21, 22, 23, 24, 25, 26, 28, 29, 30]	2.5
XuanYuan-6B + Flexora	Hellaswag	[1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 13, 14, 17]	1.7
	PIQA	[3, 4, 7, 8, 12, 14, 16, 17, 19, 21, 23, 25, 28, 29]	1.8
	RACE	[0, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 17, 20, 21, 22, 25, 28, 29]	2.5
	Winogrande	[2, 3, 4, 8, 9, 10, 14, 15, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]	2.5
Qwen1.5-7B + Flexora	Hellaswag	[0, 1, 2, 3, 4, 5, 6, 7, 9, 17]	1.3
	PIQA	[0, 1, 2, 3, 4, 5, 6, 7, 8, 11, 13, 14, 15, 17]	1.8
	RACE	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]	1.7
	Winogrande	[0, 1, 2, 3, 4, 5, 6, 7, 8, 21, 24, 25, 27, 28, 30]	2.0

Table 14: Detailed display of selected layers in the first ablation study. This table presents the results of the first ablation experiment, where the number of layers selected by Flexora was kept constant, but different layers were chosen for fine-tuning. The table includes three different random layer selection strategies (Random1, Random2, and Random3) applied to various datasets (HellaSwag, PIQA, RACE, and Winogrande). For each random selection method, the "Layer selection" column lists the indices of the layers chosen for fine-tuning in each dataset. The "Parameter(M)" column indicates the total number of parameters (in millions) used in each configuration. This detailed breakdown allows for a clear understanding of how different layer selection strategies impact the performance of model across different datasets while maintaining a consistent number of layers for fine-tuning.

Methods	Dataset	Layer selection	Parameter(M)
Random1	Hellaswag	[0, 1, 2, 3, 4, 6, 7, 8, 10, 11, 14, 18, 19, 20, 21, 25, 26, 27, 28]	2.0
	PIQA	[0, 2, 4, 10, 12, 16, 17, 18, 23, 24, 25, 26, 27, 28, 29, 30]	1.7
	RACE	[1, 2, 4, 7, 9, 11, 12, 14, 15, 18, 20, 23, 24, 26, 28, 30]	1.7
	Winogrande	[1, 2, 4, 5, 9, 10, 11, 13, 15, 17, 20, 21, 24, 26, 30, 31]	1.7
Random2	Hellaswag	[0, 2, 3, 4, 5, 6, 10, 12, 13, 15, 17, 20, 21, 22, 23, 24, 28, 29, 30]	2.0
	PIQA	[0, 1, 3, 4, 8, 13, 14, 18, 19, 22, 24, 26, 28, 29, 30, 31]	1.7
	RACE	[5, 6, 7, 8, 9, 11, 12, 13, 15, 19, 20, 21, 25, 27, 28, 30]	1.7
	Winogrande	[2, 5, 6, 7, 8, 10, 11, 13, 14, 17, 18, 22, 25, 26, 28, 30]	1.7
Random3	Hellaswag	[0, 1, 3, 4, 6, 9, 12, 13, 17, 18, 19, 20, 21, 22, 25, 26, 27, 28, 29]	2.0
	PIQA	[0, 3, 4, 9, 12, 13, 14, 15, 16, 24, 25, 26, 27, 28, 30, 31]	1.7
	RACE	[0, 1, 2, 9, 11, 12, 14, 18, 19, 20, 21, 23, 25, 26, 29, 30]	1.7
	Winogrande	[2, 4, 6, 8, 10, 12, 14, 16, 17, 18, 20, 22, 23, 29, 30, 31]	1.7

Table 15: Detailed display of layer selection with varying numbers of searching samples. This table presents the results of an experiment where different numbers of searching samples (1000, 2000, 5000, and 10000) were used to determine the layers for Flexora. The datasets involved in this experiment include HellaSwag, PIQA, RACE, and Winogrande. For each number of searching samples, the "Layer selection" column lists the indices of the layers chosen for fine-tuning in each dataset. The "Parameter(M)" column indicates the total number of parameters (in millions) used in each configuration. This detailed breakdown provides insights into how the number of searching samples impacts the layer selection process and the performance of model across different datasets.

Methods	Dataset	Layer selection	Parameter(M)
1000 searching samples	Hellaswag	[0, 2, 4, 5, 6, 8, 10, 16, 21, 26, 27, 28, 30, 31]	1.5
	PIQA	[0, 1, 2, 3, 4, 16, 25, 26, 27, 28, 29, 30, 31]	1.4
	RACE	[0, 1, 2, 3, 4, 16, 21, 28, 29, 30, 31]	1.2
	Winogrande	[0, 1, 2, 3, 4, 16, 20, 25, 26, 27, 28, 29, 30, 31]	1.5
2000 searching samples	Hellaswag	[1, 2, 3, 4, 8, 10, 11, 16, 30, 31]	1.0
	PIQA	[0, 1, 2, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]	1.5
	RACE	[0, 1, 2, 3, 4, 10, 20, 23, 27, 28, 29, 30, 31]	1.4
	Winogrande	[0, 1, 2, 3, 4, 20, 25, 27, 30, 31]	1.0
5000 searching samples	Hellaswag	[0, 1, 2, 3, 4, 8, 31]	0.7
	PIQA	[0, 2, 3, 4, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]	1.6
	RACE	[1, 3, 4, 6, 9, 10, 11, 12, 14, 27, 28, 29, 30, 31]	1.5
	Winogrande	[1, 2, 3, 4, 6, 7, 8, 9, 26, 27, 30, 31]	1.3
10000 searching samples	Hellaswag	[0, 1, 4, 10, 12, 14, 21, 24, 26, 27, 28, 29, 30, 31]	1.5
	PIQA	[0, 1, 3, 4, 7, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31]	1.7
	RACE	[1, 2, 7, 13, 14, 23, 25, 26, 27, 28, 29, 31]	1.3
	Winogrande	[6, 7, 9, 10, 15, 19, 20, 22, 26, 27, 30, 31]	1.3

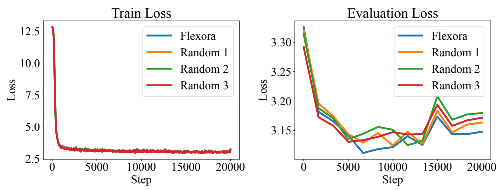

Figure 6: Comparison of train loss and evaluation loss in the Hellaswag dataset during the first ablation study. This figure presents the train loss (left) and evaluation loss (right) over 20,000 steps for the Hellaswag dataset, where the batch size at each step is 1. The performance of the Flexora method is compared against three different random layer selection strategies (Random 1, Random 2, and Random 3). The train loss graph shows how the training performance of model evolves, while the evaluation loss graph highlights the generalization capability of model on the validation set. This detailed comparison provides insights into the effectiveness of Flexora relative to random selection methods in terms of both training and evaluation metrics.

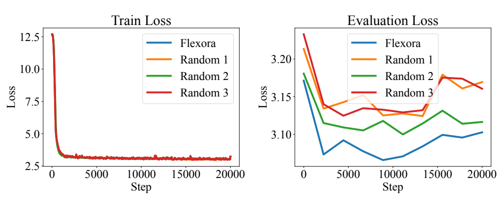

Figure 7: Comparison of train loss and evaluation loss in the PIQA dataset during the first ablation study. This figure presents the train loss (left) and evaluation loss (right) over 20,000 steps for the PIQA dataset, where the batch size at each step is 1. The performance of the Flexora method is compared against three different random layer selection strategies (Random 1, Random 2, and Random 3). The train loss graph shows how the training performance of model evolves, while the evaluation loss graph highlights the generalization capability of model on the validation set. This detailed comparison provides insights into the effectiveness of Flexora relative to random selection methods in terms of both training and evaluation metrics.

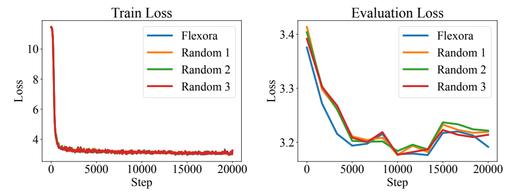

Figure 8: Comparison of train loss and evaluation loss in the RACE dataset during the first ablation study. This figure presents the train loss (left) and evaluation loss (right) over 20,000 steps for the RACE dataset, where the batch size at each step is 1. The performance of the Flexora method is compared against three different random layer selection strategies (Random 1, Random 2, and Random 3). The train loss graph shows how the training performance of model evolves, while the evaluation loss graph highlights the generalization capability of model on the validation set. This detailed comparison provides insights into the effectiveness of Flexora relative to random selection methods in terms of both training and evaluation metrics.

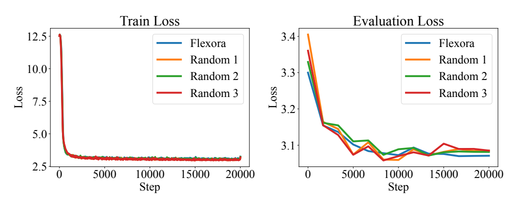

Figure 9: Comparison of train loss and evaluation loss in the Winogrande dataset during the first ablation study. This figure presents the train loss (left) and evaluation loss (right) over 20,000 steps for the Winogrande dataset, where the batch size at each step is 1. The performance of the Flexora method is compared against three different random layer selection strategies (Random 1, Random 2, and Random 3). The train loss graph shows how the training performance of model evolves, while the evaluation loss graph highlights the generalization capability of model on the validation set. This detailed comparison provides insights into the effectiveness of Flexora relative to random selection methods in terms of both training and evaluation metrics.

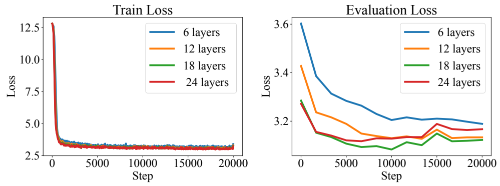

Figure 10: Training loss and evaluation loss during fine-tuning of different numbers of layers in the Flexora on the Hellaswag dataset. This figure presents the training loss (left) and evaluation loss (right) over 20,000 steps for the Hellaswag dataset. The performance is compared across four different configurations where the first 6, 12, 18, and 24 layers of the Flexora model are fine-tuned. The training loss graph shows that the model with 24 layers (red) achieves the lowest training loss, indicating it fits the training data very well. However, the evaluation loss graph reveals that the model with 18 layers (green) achieves the lowest evaluation loss, suggesting better generalization to unseen data. This discrepancy highlights the overfitting issue, where the model with 24 layers performs well on the training data but does not generalize as effectively as the model with 18 layers.

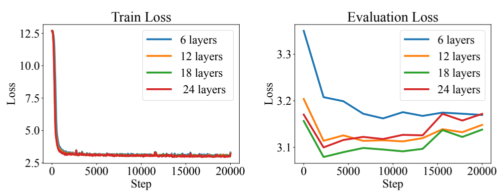

Figure 11: Training loss and evaluation loss during fine-tuning of different numbers of layers in the Flexora on the PIQA dataset. This figure presents the training loss (left) and evaluation loss (right) over 20,000 steps for the PIQA dataset. The performance is compared across four different configurations where the first 6, 12, 18, and 24 layers of the Flexora model are fine-tuned. The training loss graph shows that the model with 24 layers (red) achieves the lowest training loss, indicating it fits the training data very well. However, the evaluation loss graph reveals that the model with 18 layers (green) achieves the lowest evaluation loss, suggesting better generalization to unseen data. This discrepancy highlights the overfitting issue, where the model with 24 layers performs well on the training data but does not generalize as effectively as the model with 18 layers.

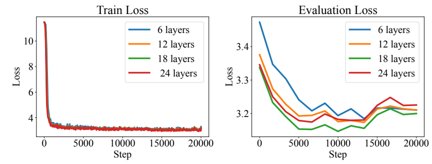

Figure 12: Training loss and evaluation loss during fine-tuning of different numbers of layers in the Flexora on the RACE dataset. This figure presents the training loss (left) and evaluation loss (right) over 20,000 steps for the RACE dataset. The performance is compared across four different configurations where the first 6, 12, 18, and 24 layers of the Flexora model are fine-tuned. The training loss graph shows that the model with 24 layers (red) achieves the lowest training loss, indicating it fits the training data very well. However, the evaluation loss graph reveals that the model with 18 layers (green) achieves the lowest evaluation loss, suggesting better generalization to unseen data. This discrepancy highlights the overfitting issue, where the model with 24 layers performs well on the training data but does not generalize as effectively as the model with 18 layers.

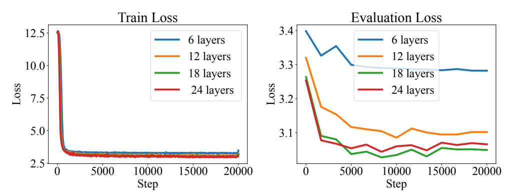

Figure 13: Training loss and evaluation loss during fine-tuning of different numbers of layers in the Flexora on the Winogrande dataset. This figure presents the training loss (left) and evaluation loss (right) over 20,000 steps for the Winogrande dataset. The performance is compared across four different configurations where the first 6, 12, 18, and 24 layers of the Flexora model are fine-tuned. The training loss graph shows that the model with 24 layers (red) achieves the lowest training loss, indicating it fits the training data very well. However, the evaluation loss graph reveals that the model with 18 layers (green) achieves the lowest evaluation loss, suggesting better generalization to unseen data. This discrepancy highlights the overfitting issue, where the model with 24 layers performs well on the training data but does not generalize as effectively as the model with 18 layers.

E Special cases

This section details the performance of Flexora and LoRA across four distinct datasets. The results indicate that Flexora demonstrates superior comprehension and judgment on more challenging questions within the test dataset, compared to LoRA. In certain instances, Flexora successfully explains problems not previously encountered during training, showcasing its robust learning and generalization capabilities.

Special cases of Hellaswag

```
dataset: Hellaswag
    "1": {
        "origin_prompt": "A lady walks to a barbell. She bends down and grabs
        the pole. The lady\n
        Question: Which ending makes the most sense?\n
        A. swings and lands in her arms.\n
        B. pulls the barbell forward.\n
        C. pulls a rope attached to the barbell.\n
        D. stands and lifts the weight over her head.\n
        You may choose from 'A', 'B', 'C', 'D'.\n
        Answer:",
        "Flexora prediciton": " D",
        "LoRA prediciton" : "B",
        "gold": "D"
    },
    "2": {
        "origin_prompt": "Two women in a child are shown in a canoe while a man
        pulls the canoe while standing in the water, with other individuals
        visible in the background. The child and a different man\n
        Question: Which ending makes the most sense?\n
        A. are then shown paddling down a river in a boat while a woman talks.\n
        B. are driving the canoe, they go down the river flowing side to side.\n
        C. sit in a canoe while the man paddles.\n
        D. walking go down the rapids, while the man in his helicopter almost
        falls and goes out of canoehood.\n
        You may choose from 'A', 'B', 'C', 'D'.\n
        Answer:",
        "Flexora prediciton": " C",
        "LoRA prediciton" : "B",
        "gold": "C"
    },
    "3": {
        "origin_prompt": "The boy lifts his body above the height of a pole.
        The boy lands on his back on to a red mat. The boy\n
        Question: Which ending makes the most sense?\n
        A. turns his body around on the mat.\n
        B. gets up from the mat.\n
        C. continues to lift his body over the pole.\n
        D. wiggles out of the mat.\n
        You may choose from 'A', 'B', 'C', 'D'.\n
        Answer:",
        "Flexora prediciton": " B",
        "LoRA prediciton" : "B",
        "gold": "B"
    }
    "4": {
        "origin_prompt": "We see a person holding face wash then putting it on
        their face. They rinse the face and add the face wash with a brush. We\n
        Question: Which ending makes the most sense?\n
        A. see a closing title screen.\n
        B. see a black screen with the credits.\n
        C. see an illustration on how to add the wash using a brush.\n
        D. then see a replay then the person putting the face wash on.\n
        You may choose from 'A', 'B', 'C', 'D'.\n
        Answer:",
        "Flexora prediction": " C",
        "LoRA prediciton" : "A",
        "gold": "C"
    },
```

Special cases of PIQA


```
dataset: PIQA
    "1": {
        "origin_prompt": "ice box\n
        A. will turn into a cooler if you add water to it\n
        B. will turn into a cooler if you add soda to it\n
        Answer:",
        "Flexora prediciton": "A",
        "LoRA prediciton" : "A",
        "gold": "A"
    },
    "2": {
        "origin_prompt": "how do you put eyelashes on?\n
        A. glue them on with mascara.\n
        B. put eyelash glue on the fake eyelashes and then let it get tacky.
        then, place it on top of your actual eyelashes and let it dry on.\n
        Answer:",
        "Flexora prediciton": "A",
        "LoRA prediciton" : "B",,
        "gold": "B"
    },
    "3": {
        "origin_prompt": "How do I fill holes and tiny gaps in the concrete when
        making a concrete countertop?\n
        A. Use  a concrete slurry\n
        B. Use  a concrete brush\n
        Answer:",
        "Flexora prediciton": "A",
        "LoRA prediciton" : "B",
        "gold": "A"
    }
    "4": {
        "origin_prompt": "When I'm deep frying a turkey, how defrosted
        should it be?\n
        A. It should be completely defrosted, otherwise the oil may
        boil over and start a fire.\n
        B. It should be completely frozen, otherwise the oil may boil
        over and start a fire.\n
        Answer:",
        "\ours{} prediction": " A\nExplanation: A turkey should be completely
        defrosted before deep frying. If it is not, the oil may boil over
        and start a fire.",
        "Flexora prediciton" : "A",
        "LoRA prediciton" : "B",
        "gold": "A"
    },
    "5": {
        "origin_prompt": "How do you properly prepare a steak.\n
        A. Take the steak out of warm storage and let come to room temperature,
        generously add salt and pepper to both sides and let sit
        for 10 minutes.\n
        B. Take the steak out of cold storage and let come to room temperature,
        generously add salt and pepper to both sides and let sit
        for 10 minutes.\n
        Answer:",
        "Flexora prediction": " B\nExplanation: B. Taking the steak
        out of cold storage and letting it come to room temperature is
        the correct answer. The steak should be at room temperature before
        cooking. The steak should be generously salted and peppered on both
        sides and let sit for 10 minutes.",
        "LoRA prediciton" : "B",
        "gold": "B"
    },
    "6": {
        "origin_prompt": "To cream butter and sugar together, you can\n
        A. Place it in a bowl and use a hand warmer\n
        B. Place in a bowl and use a hand mixer\n
        Answer:",
        "Flexora prediction": " B\nExplanation: B. Place in a bowl and
        use a hand mixer\nExplanation: To cream butter and sugar together,
        you can place it in a bowl and use a hand mixer.",
        "LoRA prediciton" : "B",
        "gold": "B"
    },
```

Special cases of RACE


```
dataset: RACE
    "1": {
        "origin_prompt": "Read the article, and answer the question by replying A,
        B, C or D.\n\n
        Article:\nThe rain had continued for a week and the flood
        had created a big river which were running by Nancy Brown's
        farm. As she tried to gather her cows to a higher ground,
        she slipped and hit her head on a fallen tree trunk.
        The fall made her unconscious for a moment or two. When she came to,
        Lizzie, one of her oldest and favorite cows, was licking her face. \n
        At that time, the water level on the farm was still rising.
        Nancy gathered all her strength to get up and began walking
        slowly with Lizzie. The rain had become much heavier,
        and the water in the field was now waist high. Nancy's pace
        got slower and slower because she felt a great pain in her head.
        Finally, all she could do was to throw her arm around Lizzie's
        neck and try to hang on. About 20 minutes later, Lizzie managed
        to pull herself and Nancy out of the rising water and onto
        a bit of high land, which seemed like a small island in
        the middle of a lake of white water. \n
        Even though it was about noon, the sky was so dark and the rain
        and lightning was so bad that it took rescuers more than
        two hours to discover Nancy. A man from a helicopter
        lowered a rope, but Nancy couldn't catch it. A moment later,
        two men landed on the small island from a ladder in the helicopter.
        They raised her into the helicopter and took her to the school gym,
        where the Red Cross had set up an emergency shelter.
        \nWhen the flood disappeared two days later, Nancy immediately
        went back to the \"island.\" Lizzie was gone. She was one of
        19 cows that Nancy had lost in the flood. \"I owe my life to
        her,\" said Nancy with tears.\n\n
        Q: What did Nancy try to do before she fell over?\n\n
        A. Measure the depth of the river\n
        B. Look for a fallen tree trunk\n
        C. Protect her cows from being drowned\n
        D. Run away from the flooded farm\n",
",
        "Flexora prediciton": "D",
        "LoRA prediciton" : "B",
        "gold": "D"
    }
```

Special cases of Winogrande


```
dataset: Winogrande
     "1": {
        "origin_prompt": "Question: Sarah was a much better surgeon
        than Maria so _ always got the easier cases.\n
        A. Sarah\n
        B. Maria\n
        Answer:",
        "Flexora prediciton": "B",
        "LoRA prediciton" : "B",
        "gold": "B"
    },
    "2": {
        "origin_prompt": "Question: Sarah was a much better surgeon
        than Maria so _ always got the harder cases.\n
        A. Sarah\n
        B. Maria\n
        Answer:",
        "Flexora prediciton": "B",
        "LoRA prediciton" : "B",
        "gold": "A"
    },
    "3": {
        "origin_prompt": "Question: They were worried the wine would ruin
        the bed and the blanket, but the _ was't ruined.\n
        A. blanket\n
        B. bed\n
        Answer:",
        "Flexora prediciton": "B",
        "LoRA prediciton" : "A",
        "gold": "B"
    },
    "4": {
        "origin_prompt": "Question: Terry tried to bake the eggplant
        in the toaster oven but the _ was too big.\n
        A. eggplant\n
        B. toaster\nAnswer:",
        "Flexora prediction": " A\nExplanation: The eggplant was
        too big to fit in the toaster oven.",
        "LoRA prediciton" : "B",
        "gold": "A"
    },
    "5": {
        "origin_prompt": "Question: At night, Jeffrey always stays up
        later than Hunter to watch TV because _ wakes up late.\n
        A. Jeffrey\n
        B. Hunter\n
        Answer:",
        "Flexora prediciton": "A",
        "LoRA prediciton" : "B",
        "gold": "A"
    },
    "6": {
        "origin_prompt": "Question: The cat of Sarah has some mouth problems,
        so she takes it to see Maria. _ is a responsible cat owner.\n
        A. Sarah\n
        B. Maria\n
        Answer:",
        "Flexora prediction": " A\nExplanation: A is a responsible cat
        owner because she takes her cat to see a veterinarian.",
        "LoRA prediciton" : "B",
        "gold": "A"
    },
    "7": {
        "origin_prompt": "Question: Benjamin was chosen instead of Brett to
        be the makeup artist for the play because _ was less experienced.\n
        A. Benjamin\n
        B. Brett\n
        Answer:",
        "Flexora prediction": " B",
        "LoRA prediciton" : "A",
        "gold": "B"
    },
```