BeamLoRA: Beam-Constraint Low-Rank Adaptation

Anonymous ACL submission

Abstract

Due to the demand for efficient fine-tuning of large language models, Low-Rank Adaptation (LoRA) has been widely adopted as one of the most effective parameter-efficient fine-tuning methods. Nevertheless, while LoRA improves efficiency, there remains room for improvement in accuracy. Herein, we adopt a novel perspective to assess the characteristics of LoRA ranks. The results reveal that different ranks within the LoRA modules not only exhibit varying levels of importance but also evolve dynamically throughout the fine-tuning process, which may limit the performance of LoRA. Based on these findings, we propose BeamLoRA, which conceptualizes each LoRA module as a beam where each rank naturally corresponds to a potential sub-solution, and the fine-tuning process becomes a search for the optimal sub-solution combination. BeamLoRA dynamically eliminates underperforming sub-solutions while expanding the parameter space for promising ones, enhancing performance with a fixed rank. Extensive experiments across three base models and 12 datasets spanning math reasoning, code generation, and commonsense reasoning demonstrate that BeamLoRA consistently enhances the performance of LoRA, surpassing the other baseline methods.

1 Introduction

In recent years, large language models have shown tremendous potential in various applications (Touvron et al., 2023a,b; Jiang et al., 2023; OpenAI, 2023). To further enhance model performance on specific downstream tasks, fine-tuning is usually the most effective approach. However, as the scale of models keeps increasing, fine-tuning all model parameters becomes unsustainable. To address this issue, parameter-efficient fine-tuning (PEFT) emerges as a practical solution (Houlsby et al., 2019; Li and Liang, 2021; Liu et al., 2022; Hu et al., 2022). By updating only lightweight modules, these methods nearly achieve the results of full parameter fine-tuning while reducing both fine-tuning time and memory usage.

Among these PEFT methods, Low-Rank Adaptation (LoRA) stands out for its effectiveness and practicality (Hu et al., 2022). The method strategically inserts trainable low-rank modules into frozen linear layers, approximating weight updates while preserving the original model architecture and inference efficiency. Recent advancements aim to enhance LoRA through various approaches: DoRA (Liu et al., 2024) decouples the fine-tuning process into directional and magnitude adjustments, whereas AdaLoRA (Zhang et al., 2023b) and IncreLoRA (Zhang et al., 2023a) dynamically optimize rank allocation across different modules. However, when revisiting the fundamental aspects of LoRA, we find these methods generally treat rank dimensions as homogeneous units, neglecting the potential hierarchical importance of individual rank components within each LoRA module.

In this paper, we adopt a novel perspective by studying the intrinsic characteristics of LoRA ranks from both spatial and temporal dimensions. From the spatial dimension, we find significant differences in the importance of ranks within a LoRA module, and pruning the less important ranks has a minimal impact on performance. From the temporal dimension, these important differences do not show up at the beginning of fine-tuning, but gradually emerge and stabilize as the fine-tuning process progresses. Despite the significant differences in importance among ranks, existing works typically allocate the same parameter budget to each rank (i.e., a row and a column in a module), which leads to constrained optimization space for important ranks and wasted resources on less important ones.

Based on the spatial and temporal findings, we propose BeamLoRA, which is inspired by beam search (Lowerre and Reddy, 1976) and treats each LoRA module as a beam, where each rank acts as a sub-solution, and the fine-tuning process is for-

malized as searching for the optimal combination of sub-solutions. Specifically, the main process of BeamLoRA includes assessment, pruning, and expansion. To assess the importance of each sub-solution, we insert a trainable score vector into the low-rank subspace and integrate the assessment process into fine-tuning. Based on their importance, we prune unimportant sub-solutions to free up space and expand the important ones, thereby allowing them to be better optimized. Furthermore, to better determine the pruning or expansion threshold, we introduce a dynamic Top-P method that achieves adaptability in both temporal and spatial dimensions. Through these mechanisms, BeamLoRA can effectively allocate parameter capacity to the most promising solution paths.

We validate our approach using three different base models across 12 datasets covering math reasoning, code generation, and commonsense reasoning. Results indicate that BeamLoRA consistently outperforms multiple LoRA-enhanced baselines. Notably, on the most challenging math reasoning and code generation tasks, BeamLoRA achieves a 1.57% accuracy gain while using only 2.4% of the trainable parameters compared to full fine-tuning. Further analysis reveals that the success of BeamLoRA is attributed to its increased important rank space within the LoRA module.

In summary, our contributions are as follows:

- We adopt a novel perspective by studying the characteristics of LoRA ranks from both spatial and temporal dimensions, and highlight that ranks with various importance are assigned an equally sized parameter space.
- We introduce BeamLoRA and view a LoRA module as a beam. It continuously assesses the importance of each rank, compresses the less important ones, and frees up resources for the more significant ones.
- Through extensive experiments across three base models of different sources and scales, along with 12 diverse datasets, we demonstrate that BeamLoRA consistently outperforms other baselines.

2 Preliminary

2.1 Low-Rank Adaptation (LoRA)

Considering that the updates for fine-tuning large models occur within a low-rank subspace (Aghajanyan et al., 2021), LoRA inserts low-rank modules into the linear layers of the base model to approximate these transformations. Specifically, for a weight matrix $\mathbf{W}_0 \in \mathbb{R}^{d\times k}$, LoRA decomposes the update $\Delta\mathbf{W}$ into a low-rank matrices product $\mathbf{BA}$, where $\mathbf{B} \in \mathbb{R}^{d\times r}$, $\mathbf{A} \in \mathbb{R}^{r\times k}$, and $r \ll min(d, k)$. The forward pass of LoRA is formulated as

$$y = \mathbf{W}_0 x + \Delta\mathbf{W}x = \mathbf{W}_0 x + \mathbf{BA}x, \quad (1)$$

where $x \in \mathbb{R}^d$ represents the input and $y \in \mathbb{R}^d$ is the output. During fine-tuning, $\mathbf{W}_0$ remains frozen, while only $\mathbf{B}$ and $\mathbf{A}$ matrices are trainable.

The Independence of Ranks. Given a LoRA module that includes two matrices $\mathbf{B}$ and $\mathbf{A}$, in which $\mathbf{B}$ is represented as $[\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_r]$, where $\mathbf{b}_i$ denotes the i-th column of matrix $\mathbf{B}$, and $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_r]$, where $\mathbf{a}_i$ denotes the i-th row. In this way, the update $\Delta\mathbf{W}$ is equivalent to

$$\begin{aligned}\Delta\mathbf{W} =& \mathbf{BA} = \begin{bmatrix}\mathbf{b}_1 & \mathbf{b}_2 & ... & \mathbf{b}_r\end{bmatrix} \begin{bmatrix}\mathbf{a}_1 \\ \mathbf{a}_2 \\ ... \\ \mathbf{a}_r\end{bmatrix}, \\ =& \mathbf{b}_1\mathbf{a}_1 + ... + \mathbf{b}_r\mathbf{a}_r = \sum_r \mathbf{b}_i\mathbf{a}_i = \sum_r \Delta\mathbf{w}_i,\end{aligned} \quad (2)$$

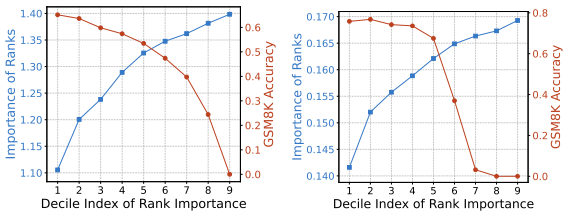
where $\Delta\mathbf{w}_i \in \mathbb{R}^{d\times k}$ represents the update matrix of i-th rank. Thus, the LoRA fine-tuning process can be viewed as independently updating each $\Delta\mathbf{w}_i$ represented by each rank.

2.2 Analysis of LoRA Ranks

During the fine-tuning process, an intuitive assumption is that each rank within a LoRA module contributes similarly. This intuition may stem from the standard LoRA initialization procedure, where matrix $\mathbf{A}$ is initialized randomly, and matrix $\mathbf{B}$ starts with zero values. Since all $\Delta\mathbf{w}_i$ matrices begin as zero matrices and are updated simultaneously, their contributions might remain comparable throughout the fine-tuning process.

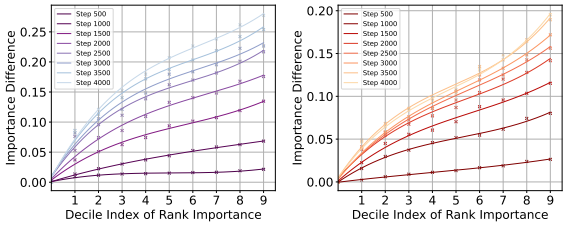
To examine the validity of this assumption, we fine-tune LoRA on LLaMA2-7B (Touvron et al., 2023b) and Mistral-7B-v0.1 (Jiang et al., 2023) with the MetaMathQA dataset (Yu et al., 2024) and conduct an analysis from both spatial and temporal dimensions. Given that LoRA updates represent adjustments to pre-trained weights, we use the magnitude of each $\Delta\mathbf{w}$ to quantify the importance of different ranks[1].

[1]The magnitude (importance) of the matrix is roughly measured by the commonly used Frobenius norm.



(a) LLaMA2-7B (b) Mistral-7B-v0.1

Figure 1: Differences in importance among ranks within a LoRA module (*spatial*). The blue line represents the deciles of importance for ranks. The red line represents accuracy changes when pruning ranks of varying importance. We take `ffn.up_proj` in the 30th layer as an example, with similar phenomena in other modules.



(a) `ffn.up_proj` (b) `attn.v_proj`

Figure 2: The visualization of importance difference among different ranks in LoRA on LLaMA2-7B as fine-tuning steps increase (*temporal*). We take `ffn.up_proj` and `attn.v_proj` in the 30th layer as examples, with similar trends observed in all other modules.

Spatial Dimension. Figure 1 describes the results of sorting the importance of ranks after fine-tuning. The blue line represents the levels of importance across different deciles after sorted by importance in a LoRA module. It can be observed that the deciles of the importance of ranks are hierarchical, indicating that **the importance of different ranks within a LoRA module is not uniform after fine-tuning.** Furthermore, by pruning the ranks in each LoRA module based on its own importance from the least to the most significant gradually (the red line), **the accuracy shows limited change when pruning the less important ranks**. On the contrary, when important ranks are pruned, the evaluation results drop sharply to zero. This phenomenon further demonstrates the significant differences in importance among the different ranks.

Temporal Dimension. To further understand the reasons behind this importance differentiation, we return to the initial assumption: since the $\Delta\mathbf{w}$ corresponding to each rank is initialized to zero, they all start with equal importance during fine-tuning. Therefore, a natural idea is to investigate how the importance of different ranks evolves during fine-tuning. Figure 2 shows the changes in importance of two LoRA modules, where **the differences in importance among ranks increase with the number of fine-tuning steps**. In other words, the less important ranks are progressively filtered out. Furthermore, **the differences in importance tend to stabilize as the number of fine-tuning steps continues to increase further**. These phenomena are prevalent across various LoRA modules.

In summary, there are significant differences in the importance of ranks in LoRA, and these differences appear and gradually increase as the fine-tuning process progresses. However, in most existing LoRA-based methods, less important ranks still occupy the same parameter budget as important ones. Here, a question is about to arise: *Could we free up space from less important ranks for more important ones to achieve better optimization?*

3 BeamLoRA

To answer the above question, we propose BeamLoRA, which continuously assesses the importance of different ranks during fine-tuning, periodically pruning the less important ones to free up resources for the more important ranks. The overall workflow of the method is illustrated in Figure 3.

For a LoRA module with rank r, we treat it as a beam with width r and the optimization process is naturally regarded as a search for the solution set $\Delta\mathbf{W} = \{\Delta\mathbf{w}_1, \Delta\mathbf{w}_2, ..., \Delta\mathbf{w}_r\}$ tailored to the fine-tuning dataset, where i-th rank in the LoRA module is considered a sub-solution $\Delta\mathbf{w}_i$. Formally, the optimization process seeks to minimize the loss function $\mathcal{L}$ over dataset $\mathcal{D}$:

$$\Delta\mathbf{W}^* = \arg\min \ \mathcal{L}(\mathbf{W}_0 + \Delta\mathbf{W};\ \mathcal{D}), \quad (3)$$

where the optimal solution $\Delta\mathbf{W}^*$ represents the well-trained LoRA module.

3.1 Importance Assessment

In the pilot experiments of Section 2.2, we use the Frobenius norm to measure the importance of each rank offline. However, this approach involves considerable computational overhead during fine-tuning[2]. To make the assessment more efficient and accurate, we introduce a learnable score vector

[2]Typically, a large model contains hundreds of LoRA modules. For each module, it requires computing r matrices of size $d \times k$, and then calculating the norm for each matrix.

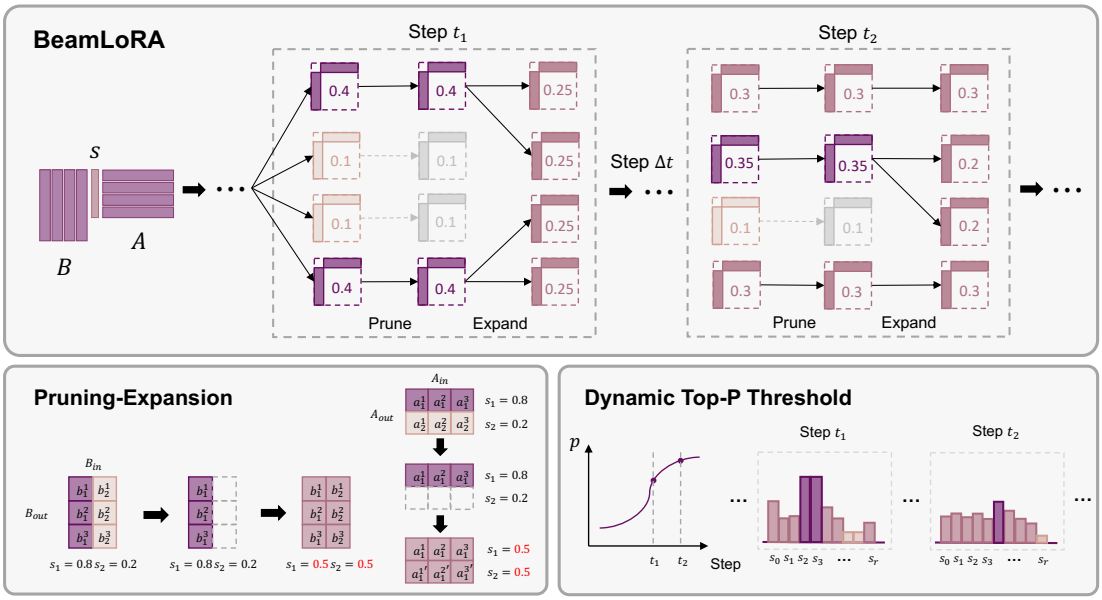


Figure 3: Illustration of BeamLoRA. Throughout the fine-tuning process, BeamLoRA continually assesses the importance of each rank. Every Δt steps, unimportant ranks are pruned while those identified as important are expanded, optimizing the module's performance.

$\mathbf{s} \in \mathbb{R}^r$, which is inserted between the matrices $\mathbf{B}$ and $\mathbf{A}$, to scale the output of each rank through element-wise broadcasting multiplication. In that case, the modified forward pass of LoRA can be formulated as follows:

$$y = \mathbf{W}_0 x + \mathbf{B}(\mathbf{s} \odot \mathbf{A})x = \sum_r s_i \Delta \mathbf{w}_i x. \tag{4}$$

This is equivalent to scaling the corresponding rank matrices $\Delta \mathbf{w}_i$. This means that during fine-tuning, if a rank is considered important, the corresponding score s_i for that rank is amplified.

At the start of the fine-tuning process, LoRA initializes each rank to zero, indicating that their initial importance is equal. Consequently, we initialize all elements in $\mathbf{s}$ with identical values. During the fine-tuning process, $\mathbf{s}$ is consistently normalized using the softmax function, like the logits of tokens in text generation,

$$s_i = \frac{e^{s_i}}{\sum_{j=1}^{r} e^{s_j}}, \tag{5}$$

where s_i is the i-th element in score vector $\mathbf{s}$. The continuous normalization ensures a stable value range and facilitates meaningful comparisons of importance differences between elements.

3.2 Pruning and Expansion

With the importance of each rank, the space occupied by the less important ranks can be freed up, which allows us to expand the parameter space for the remaining important ones. Specifically, we begin by selecting the K least important ranks based on their importance $\mathbf{s}$ to form the rank index set $\mathcal{I}_p$ for pruning:

$$\mathcal{I}_p = \{i \mid s_i \in \text{Min}_K(\mathbf{s})\}. \tag{6}$$

During the pruning stage, for the indices of the unimportant ranks included in $\mathcal{I}_p$, we set their parameters to zero:

$$\mathbf{b}_i, \mathbf{a}_i = \begin{cases} 0 & i \in \mathcal{I}_p, \\ \mathbf{b}_i, \mathbf{a}_i & \text{otherwise}, \end{cases} \tag{7}$$

where i is the index of i-th rank. It means that if i is in $\mathcal{I}_p$, we set both $\mathbf{a}_i$ and $\mathbf{b}_i$ of i-th rank to zero in preparation for subsequent expansion.

Next, more space is allocated for important ranks for better optimization. Similarly, we select the K most important ranks based on $\mathbf{s}$ to form the rank index set $\mathcal{I}_e$ for expansion:

$$\mathcal{I}_e = \{i \mid s_i \in \text{Top}_K(\mathbf{s})\}. \tag{8}$$

For each pruned rank, we copy the parameter values from the corresponding important rank:

$$\mathbf{b}_{\mathcal{I}_p}, \mathbf{a}_{\mathcal{I}_p} \leftarrow \mathbf{b}_{\mathcal{I}_e}, \mathbf{a}_{\mathcal{I}_e}. \tag{9}$$

Meanwhile, to ensure the stability of the optimization for the expanded ranks, the optimizer states are also copied:

$$\mathbf{M}_{\mathcal{I}_p}, \mathbf{V}_{\mathcal{I}_p} \leftarrow \mathbf{M}_{\mathcal{I}_e}, \mathbf{V}_{\mathcal{I}_e}, \tag{10}$$

where $\mathbf{M}$ and $\mathbf{V}$ are the first-order and second-order moment in Adam optimizer.

However, directly copying parameters and optimizer states from the original ranks creates a challenge: the lack of symmetry breaking between the expanded and original ranks means the optimization process is essentially trying to synchronously optimize two identical objects (Chen et al., 2016). This makes it difficult to effectively leverage the additional capacity provided by the expanded parameter space. To address this issue, we propose using historical parameters and their corresponding optimizer states to break the symmetry, Eq. 9 and Eq. 10 change to:

$$\mathbf{b}_{\mathcal{I}_p}, \mathbf{a}_{\mathcal{I}_p} \leftarrow \mathbf{b}'_{\mathcal{I}_e}, \mathbf{a}'_{\mathcal{I}_e}, \tag{11}$$

$$\mathbf{M}_{\mathcal{I}_p}, \mathbf{V}_{\mathcal{I}_p} \leftarrow \mathbf{M}'_{\mathcal{I}_e}, \mathbf{V}'_{\mathcal{I}_e}, \tag{12}$$

where $\mathbf{b}'_{\mathcal{I}_e}$ and $\mathbf{a}'_{\mathcal{I}_e}$ represent the historical parameters of the important ranks, $\mathbf{M}'_{\mathcal{I}_e}$ and $\mathbf{V}'_{\mathcal{I}_e}$ represent the optimizer states[3]. After expansion, we take the average of the corresponding expanded $s_{\mathcal{I}_p}$ and $s_{\mathcal{I}_e}$ to ensure fair competition between the expanded ranks and the original ones.

3.3 Dynamic Top-P Threshold

In the previous statement, we fix the number of ranks for each pruning or extension operation to be K. This might overlook the actual distribution of parameter importance, potentially leading to the elimination of relatively important parameters due to quantitative constraints. Similar to the sampling process of text generation, we introduce Top-P strategy (Holtzman et al., 2020; Huang et al., 2024) to dynamically determine the operable rank number. Specifically, given the score vector $\mathbf{s}$ and a threshold p, we sort s_i in descending order, then identify the subset of operable ranks and its size as K:

$$K = |\{i \mid \sum_{j=1}^{i} s_j \geq p\}|, \tag{13}$$

where i is the index of i-th rank. A larger p results in fewer ranks being operated, while a smaller one leads to more ranks being affected.

Even so, a fixed threshold p still poses issues, as the learning rate decreases and the model converges, the number of ranks that need to be operated should decrease. Therefore, we design a Dynamic Top-P Threshold. To gradually reduce the number of ranks being operated, the p value should progressively increase with each operation, starting from p_{init} and moving towards 1 (indicating no ranks are operated). We tie this process to the learning rate scheduler used during fine-tuning to align it with the model's learning progression. For example, given the commonly used cosine scheduler, We obtain the value of threshold p at step t by:

$$p = p_{\text{init}} + \frac{1}{2}(1 - p_{\text{init}}) \left(1 - \cos\left(\frac{\pi t}{T}\right)\right), \tag{14}$$

where T is the total fine-tuning steps. In implementation, we perform pruning and expansion operations every Δt steps, which allows the LoRA module to adapt after expansion.

3.4 Computational Efficiency

Regarding fine-tuning efficiency, BeamLoRA is similar to LoRA (more details in Appendix B.2), with a minimal addition of parameters in the form of a score vector $\mathbf{s}$. In terms of inference efficiency, $\mathbf{s}$ can be merged in the matrix $\mathbf{A}$: $\mathbf{A}' = \mathbf{s} \odot \mathbf{A}$, resulting in a structure identical to standard LoRA. Furthermore, the design philosophy of BeamLoRA ensures consistent ranks in various modules, allowing smooth integration with existing LoRA inference frameworks, which distinguishes it from previous works that employ varying ranks across different modules (Zhang et al., 2023a,b).

Note that the inspiration for BeamLoRA comes from the classic Beam Search algorithm (Lowerre and Reddy, 1976), where we consider each LoRA module as a beam. Although BeamLoRA employs similar operations, it pursues distinct objectives. Beam Search aims to produce a single sentence to achieve the final goal, resulting in only one solution. In contrast, our approach continuously filters sub-solutions to obtain an optimal collection of sub-solutions to accomplish the objective.

4 Experiments

4.1 Experimental Settings

Models and Datasets. To thoroughly evaluate our method, our experiments encompass three different base models, including LLaMA2-7B, Mistral-7B-v0.1, and LLaMA2-13B. We conduct experiments across three different domains, including math reasoning, code generation, and commonsense reasoning, utilizing a total of 12 datasets. For math reasoning, we fine-tune the models on the MetaMathQA dataset (Yu et al., 2024) and

[3] In practice, we use the parameters from half steps between the last pruning and the current pruning step.

Model	Method	#Params	Math Reasoning		Code Generation		Avg.
			GSM8K	MATH	HumanEval	MBPP	
LLaMA2-7B	Full-FT†	6738M	66.50	19.80	38.01	46.03	42.59
	LoRA	160M	66.31	19.09	39.23	43.47	42.03
	DoRA	161M	65.53	19.25	38.41	42.95	41.54
	PiSSA	160M	64.87	17.67	35.77	39.33	39.41
	MiLoRA	160M	66.19	18.45	36.79	44.62	41.51
	ReLoRA	160M	62.55	18.08	35.98	45.59	40.55
	AdaLoRA	160M	**68.04**	17.02	35.16	**46.56**	41.70
	IncreLoRA	160M	65.58	16.93	34.35	42.77	39.91
	BeamLoRA	160M	67.05	**19.39**	**43.90**	46.30	**44.16**
Mistral-7B	Full-FT†	7242M	77.70	28.20	53.86	61.73	55.37
	LoRA	168M	77.56	28.04	**54.27**	60.85	55.18
	DoRA	169M	77.86	28.14	53.46	62.08	55.39
	MiLoRA	168M	77.36	26.71	50.00	**62.88**	54.24
	AdaLoRA	168M	77.91	27.53	46.95	60.14	53.13
	BeamLoRA	168M	**78.11**	**28.28**	54.07	62.70	**55.79**

Table 1: Math reasoning and code generation results for LLaMA2-7B and Mistral-7B with $r = 64$ for all methods. On Mistral-7B, we compare the baseline methods that perform well on LLaMA. The math reasoning results for Full-FT† are derived from MetaMathQA paper (Yu et al., 2024).

evaluate them using the GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021) datasets. For code generation, we fine-tune on the Code-Feedback105K dataset (Zheng et al., 2025; Meng et al., 2024) and then evaluate using the HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) datasets. For commonsense reasoning, we fine-tune on the Commonsense170K dataset (Hu et al., 2023b) and evaluate on the BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2019), SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2019), ARC-e, ARC-c (Clark et al., 2018), and OBQA (Mihaylov et al., 2018) datasets.

Baselines. We compare BeamLoRA with eight baseline methods to validate the effectiveness of our proposed approach: Full-FT, LoRA (Hu et al., 2022), DoRA (Liu et al., 2024), ReLoRA (Lialin et al., 2024), PiSSA (Meng et al., 2024), MiLoRA (Wang et al., 2024), AdaLoRA (Zhang et al., 2023b), and IncreLoRA (Zhang et al., 2023a). More details are presented in Appendix A.

4.2 Math Reasoning and Code Generation

Table 1 presents the experiments on math reasoning and code generation, demonstrating that BeamLoRA outperforms all other baseline methods in terms of overall performance. Notably, BeamLoRA not only surpasses the original LoRA across all tasks but also achieves an average performance improvement of 1.57% compared to standard full parameter fine-tuning on LLaMA2-7B. This result is obtained while maintaining the same number of fine-tuning epochs and full data settings as the standard full parameter fine-tuning, highlighting the practicality of the BeamLoRA method.

Furthermore, we extend our experiments on Mistral-7B by comparing BeamLoRA with the baseline methods that perform well in LLaMA experiments. The results show that BeamLoRA continues to outperform all baseline methods, surpassing Full-FT by 0.42%. More notably, BeamLoRA shows improvement over Full-FT across all task metrics. This demonstrates that within a limited parameter budget, BeamLoRA can effectively achieve better optimization by expanding the parameter space of important ranks.

4.3 Commonsense Reasoning

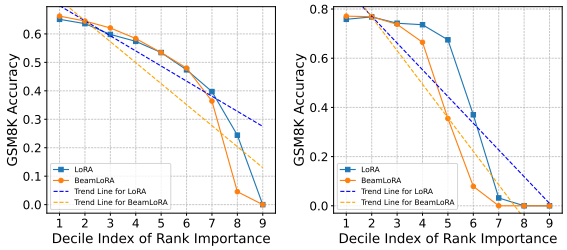
Table 2 presents evaluation results across 8 commonsense reasoning datasets. BeamLoRA achieves the best overall performance on LLaMA2-7B, with an average accuracy improvement of 3.2% over original LoRA and 1.1% over the strong baseline DoRA. Similar to math and coding tasks, BeamLoRA's performance does not rely heavily on optimal results from just two or three datasets, as observed in other baselines. Instead, it consistently

Model	Method	ARC-c	SIQA	WinoGrande	BoolQ	ARC-e	PIQA	OBQA	HellaSwag	Avg.
LLaMA2-7B	LoRA†	64.7	79.5	82.6	69.8	79.8	79.9	81.0	83.6	77.6
	DoRA†	68.2	76.0	82.6	**71.8**	83.7	83.7	82.4	89.1	79.7
	PiSSA‡	60.2	78.4	78.0	67.6	75.8	78.1	75.6	76.6	73.8
	MiLoRA‡	68.8	**80.1**	82.0	67.6	82.8	**83.8**	80.6	88.2	79.2
	ReLoRA	59.3	76.9	77.2	63.9	75.4	76.4	63.2	62.2	69.3
	AdaLoRA	69.5	66.4	78.6	62.1	**84.1**	83.2	79.2	42.1	70.7
	IncreLoRA	65.5	61.3	81.4	63.6	81.3	70.7	73.8	66.9	70.6
	BeamLoRA	**71.0**	78.9	**82.7**	71.6	83.7	82.8	**84.8**	**90.5**	**80.8**
LLaMA2-13B	LoRA	**75.8**	80.9	86.1	**75.0**	87.2	**86.2**	85.4	92.6	83.7
	DoRA	74.6	81.2	86.3	74.4	86.8	84.3	84.2	93.4	83.2
	MiLoRA	73.0	80.3	**86.7**	73.6	87.1	81.1	85.6	92.0	82.4
	AdaLoRA	**75.8**	73.1	84.5	67.7	**88.3**	83.3	83.4	90.7	80.9
	BeamLoRA	75.5	**81.3**	86.1	74.7	88.0	85.6	**86.2**	**94.3**	**84.0**

Table 2: Commonsense reasoning results for LLaMA2-7B and LLaMA2-13B. We set $r = 32$ for all methods. All results with † are taken from (Liu et al., 2024) and those marked with ‡ are taken from (Wang et al., 2024).

	GSM8K	MATH	Avg.
LoRA	66.31	19.09	42.70
BeamLoRA	67.05	19.39	43.22
w/o Expansion	65.88	18.94	42.41
w/o Assessment	64.82	19.08	41.95
w/o Dynamic P.	65.81	18.74	42.28

Table 3: Results of ablation experiments. We evaluate the impact of pruning and the significance of expansion, importance assessment, and dynamic Top-P threshold in BeamLoRA.



(a) LLaMA2-7B (b) Mistral-7B-v0.1

Figure 4: Comparison of BeamLoRA and LoRA on importance of ranks within a module by pruning ranks. The solid lines represent the accuracy changes after pruning, while the dashed lines indicate the accuracy change trends caused by pruning.

performs among the top three results across most datasets, demonstrating the generalization capability of BeamLoRA. Additionally, we find that IncreLoRA and AdaLoRA are less effective in commonsense reasoning tasks, likely due to frequent rank changes across modules, which cause instability in fine-tuning. This issue is more evident in scenarios requiring extensive task evaluation.

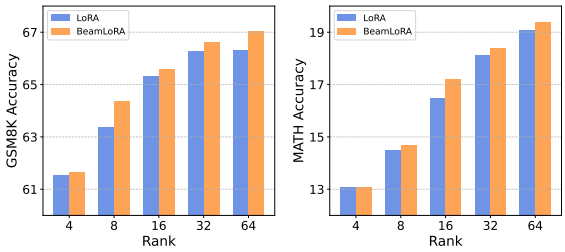
For larger base models LLaMA2-13B, the performance gaps between different methods become smaller. In this setting, BeamLoRA still achieves a 0.4% performance improvement over LoRA, while other methods show inferior performance compared to LoRA. This demonstrates that BeamLoRA can effectively enhance LoRA's performance with larger base models.

4.4 Ablation Study

The ablation results are shown in Table 3. Without expansion refers to only pruning the unimportant ranks, the performance experiences a slight decline compared to the original LoRA, and is markedly inferior to the complete BeamLoRA. This underscores the significance of expanding the important ranks. Without assessment refers to randomly pruning and expanding ranks, leading to a substantial performance drop. This suggests that important ranks may have been pruned, highlighting the necessity of rank assessment. Without dynamic Top-P refers to using a static operation threshold throughout fine-tuning, which also results in a performance decline. This indicates that higher thresholds should be applied during the later stages of fine-tuning, resulting in fewer ranks being operated. This approach allows the model to better adapt pruning and expansion as it converges, emphasizing the importance of dynamic thresholds.

4.5 Analysis

Why is BeamLoRA effective? To further understand how BeamLoRA affects the ranks within LoRA modules, building upon our observations in Figure 1, we analyze the differences in rank impor-



(a) GSM8K (b) MATH

Figure 5: Comparison of BeamLoRA and LoRA on accuracy under various rank settings on LLaMA2-7B.

tance between BeamLoRA and LoRA. As shown in Figure 4, when pruning ranks based on their decile importance within each module, we observe that BeamLoRA's accuracy decreases more rapidly compared to LoRA. This indicates that in BeamLoRA, the importance of different ranks within each module is more evenly distributed. Compared to LoRA, the number of important ranks increases in the BeamLoRA module, with each rank contributing more uniformly to the overall performance. This more balanced utilization of ranks explains why BeamLoRA consistently outperforms LoRA across various experimental settings.

How does BeamLoRA perform under different rank settings? As shown in Figure 5. We see that BeamLoRA improves the performance of LoRA across each rank setting, demonstrating the effectiveness of BeamLoRA's approach to compress unimportant ranks and expand important ones. In scenarios with very small ranks (e.g., $r = 4$), the performance improvement brought by BeamLoRA is relatively limited compared to larger rank settings. This is because, with small rank settings and difficult tasks (e.g., Math Reasoning), LoRA is denser, leaving fewer unimportant ranks to compress and expand, thus providing a smaller operational space for BeamLoRA.

5 Related Works

5.1 LoRA and its variants

As one of the parameter-efficient fine-tuning methods, LoRA (Hu et al., 2022) has been widely adopted. However, it still has room for improvement in terms of accuracy. Current enhancements follow two main pathways: optimizing initialization and refining the fine-tuning process. For initialization, methods like PiSSA (Meng et al., 2024) and MiLoRA (Wang et al., 2024) use Singular Value Decomposition on base model weights, with PiSSA focusing on principal singular values and MiLoRA on minor ones for initializing LoRA before fine-tuning. For fine-tuning, DoRA (Liu et al., 2024) splits LoRA's fine-tuning into magnitude and direction components. ReLoRA (Lialin et al., 2024) continuously merges the fine-tuned LoRA modules into the base model. AdaLoRA (Zhang et al., 2023b) and IncreLoRA (Zhang et al., 2023a) optimize rank allocation across modules. Unlike these approaches, BeamLoRA revisits the foundational aspects of LoRA and recognizes the varying importance of ranks within a module. It compresses less important ranks to free up space for expanding the important ones, thereby allowing them to be better optimized.

5.2 Model Pruning and Expansion

Model pruning is typically used to remove redundant parameters in models, thereby improving efficiency (Kurtic et al., 2022; Ma et al., 2023). Unlike previous works, our primary goal for pruning is to free up space for expanding important parameters. Model Width expansion is first introduced by Net2Net (Chen et al., 2016) and applied to CNNs. bert2BERT (Chen et al., 2022) extends this method to the pre-training of language models, and the recent work Scaling Smart (Samragh et al., 2024) applies width expansion to large scale base models. Unlike these approaches, we focus on parameter-efficient fine-tuning and propose compressing unimportant parameters within a limited space to expand important ones for better performance. Additionally, due to the shorter nature of the fine-tuning process than pre-training, we propose to use historical states to break symmetry in expansion, thereby ensuring fast convergence.

6 Conclusion

This paper introduces a PEFT method called BeamLoRA. We adopt a novel perspective to study the characteristics of ranks within a LoRA module and discover that there are differences in the importance of ranks, which change with the number of fine-tuning steps. Based on this, we propose using a dynamic threshold to prune less important ranks, free up space to better optimize the more important ones. Extensive experiments demonstrate that BeamLoRA effectively enhances the performance of LoRA across different base models, tasks, and settings, outperforming other baseline methods.

Limitations

BeamLoRA introduces a method that compresses less important ranks while expanding important ones during fine-tuning. Although this approach achieves good performance in parameter-efficient fine-tuning, existing implementation requires the addition of a trainable assessment vector $\mathbf{s}$ over ranks. In the context of full-parameter training, a full-rank matrix does not have the low-rank structure like the $\mathbf{B}$-$\mathbf{A}$ decomposition in LoRA, making it impossible to add the vector $\mathbf{s}$. How to extend this method to full-parameter training scenarios remains an area for future research exploration.

References

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2021. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7319–7328, Online. Association for Computational Linguistics.

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *CoRR*, abs/2108.07732.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2019. PIQA: reasoning about physical commonsense in natural language. *CoRR*, abs/1911.11641.

Cheng Chen, Yichun Yin, Lifeng Shang, Xin Jiang, Yujia Qin, Fengyu Wang, Zhi Wang, Xiao Chen, Zhiyuan Liu, and Qun Liu. 2022. bert2BERT: Towards reusable pretrained language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2134–2148, Dublin, Ireland. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Tianqi Chen, Ian J. Goodfellow, and Jonathon Shlens. 2016. Net2net: Accelerating learning via knowledge transfer. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

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

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the AI2 reasoning challenge. *CoRR*, abs/1803.05457.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. *CoRR*, abs/2103.03874.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Zhiqiang Hu, Yihuai Lan, Lei Wang, Wanyu Xu, Ee-Peng Lim, Roy Ka-Wei Lee, Lidong Bing, and Soujanya Poria. 2023a. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Lee. 2023b. LLM-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5254–5276, Singapore. Association for Computational Linguistics.

Quzhe Huang, Zhenwei An, Nan Zhuang, Mingxu Tao, Chen Zhang, Yang Jin, Kun Xu, Kun Xu, Liwei Chen, Songfang Huang, and Yansong Feng. 2024. Harder task needs more experts: Dynamic routing in MoE models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12883–12895, Bangkok, Thailand. Association for Computational Linguistics.

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. 2023. Mistral 7b. *CoRR*, abs/2310.06825.

Eldar Kurtic, Daniel Campos, Tuan Nguyen, Elias Frantar, Mark Kurtz, Benjamin Fineran, Michael Goin, and Dan Alistarh. 2022. The optimal BERT surgeon: Scalable and accurate second-order pruning for large language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 4163–4181, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. 2024. ReloRA: High-rank training through low-rank updates. In *The Twelfth International Conference on Learning Representations*.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. DoRA: Weight-decomposed low-rank adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 32100–32121. PMLR.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 61–68, Dublin, Ireland. Association for Computational Linguistics.

Bruce P Lowerre and B Raj Reddy. 1976. Harpy, a connected speech recognition system. *The Journal of the Acoustical Society of America*, 59(S1):S97–S97.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. LLM-pruner: On the structural pruning of large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. 2024. PiSSA: Principal singular values and singular vectors adaptation of large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391, Brussels, Belgium. Association for Computational Linguistics.

OpenAI. 2023. Gpt-4 technical report. *ArXiv*, abs/2303.08774.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. WINOGRANDE: an adversarial winograd schema challenge at scale. *CoRR*, abs/1907.10641.

Mohammad Samragh, Seyed-Iman Mirzadeh, Keivan Alizadeh-Vahid, Fartash Faghri, Minsik Cho, Moin Nabi, Devang Naik, and Mehrdad Farajtabar. 2024. Scaling smart: Accelerating large language model pre-training with small model initialization. *CoRR*, abs/2409.12903.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. 2019. Social IQa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4463–4473, Hong Kong, China. Association for Computational Linguistics.

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. 2023a. Llama: Open and efficient foundation language models. *ArXiv*, abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Hanqing Wang, Yixia Li, Shuo Wang, Guanhua Chen, and Yun Chen. 2024. Milora: Harnessing minor singular components for parameter-efficient llm finetuning. *Preprint*, arXiv:2406.09044.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng YU, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2024. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy. Association for Computational Linguistics.

Feiyu Zhang, Liangzhi Li, Junhao Chen, Zhouqiang Jiang, Bowen Wang, and Yiming Qian. 2023a. Increlora: Incremental parameter allocation method for parameter-efficient fine-tuning. *CoRR*, abs/2308.12043.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023b. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*.

Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue. 2025. Opencodeinterpreter: Integrating code generation with execution and refinement. *Preprint*, arXiv:2402.14658.

A Experimental Setup

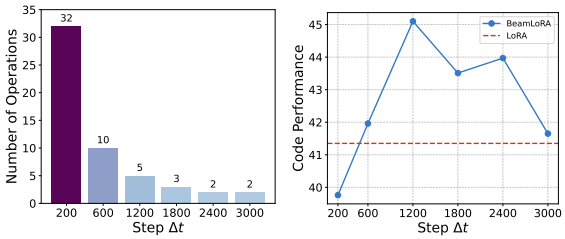
A.1 Baselines

We select several baselines to verify the effectiveness of our method. **Full-FT** fine-tunes all model parameters, delivers strong performance but requires substantial computational resources. **LoRA** (Hu et al., 2022) is one of the most efficient PEFT methods, offering computational efficiency, though its accuracy often differs from full parameter fine-tuning. **DoRA** (Liu et al., 2024) decomposes LoRA's fine-tuning into magnitude and direction components. **ReLoRA** (Lialin et al., 2024) continuously merges the obtained LoRA parameters into the base model during fine-tuning. **PiSSA** (Meng et al., 2024) and **MiLoRA** (Wang et al., 2024) perform Singular Value Decomposition (SVD) on the base model; PiSSA fine-tunes the significant components of the decomposition, while MiLoRA fine-tunes the minor components. **AdaLoRA** (Zhang et al., 2023b) begins fine-tuning with a rank setting higher than the target and prunes redundant ranks during the process to achieve optimal rank allocation across different modules. In contrast, **IncreLoRA** (Zhang et al., 2023a) starts with a rank setting lower than the target and progressively increases the rank during fine-tuning to achieve optimal rank allocation across modules.

A.2 Implementation Details

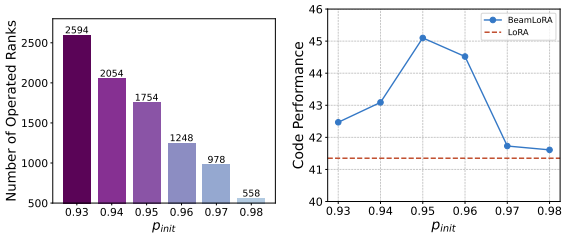
In the math and code tasks, we follow Yu et al. (2024) and set the Full-FT learning rate for LLaMA2-7B to 2e-5 and for Mistral-7B to 5e-6, with a batch size of 128. The models are fine-tuned for 3 epochs on the dataset. For the ReLoRA method, we use the same settings as Full-FT. For other PEFT methods, we add the LoRA module to all linear layers. Following Wang et al. (2024) we set their learning rate to 3e-4, a batch size of 32, and fine-tuning for 3 epochs.

In the commonsense reasoning tasks, to facilitate comparison, we follow the approach of Hu et al. (2023a) and Liu et al. (2024) by adding LoRA to q_proj, k_proj, v_proj, up_proj, and down_proj modules. We use the optimal learning rate from {2e-4,3e-4} for different methods on LLaMA2-7B, with a batch size of 16 and fine-tuning for 3 epochs. For the larger LLaMA2-13B, following Liu et al. (2024), we reduce the learning rate to the optimal {1e-4,2e-4,3e-4} for different methods, while keeping other settings unchanged. These settings are applied to all PEFT methods. All our experiments



(a) Number of Operations (b) Code Performance

Figure 6: Effect of step Δt. We set p_{init} to 0.95 and take the average performance across the code tasks on LLaMA2-7B with $r = 64$.



(a) Number of Operated Ranks (b) Code Performance

Figure 7: Effect of p_{init}. We set the step Δt to 1200 and take the average performance across the code tasks on LLaMA2-7B with $r = 64$.

are conducted on four H800 GPUs.

The implementation of BeamLoRA mainly includes assessment, pruning-expansion, and dynamic Top-P threshold. The most crucial pruning-expansion is uniformly set in the first two epochs to facilitate. BeamLoRA introduces two hyperparameters: the initial value of the dynamic threshold p_{init} and the operation interval step Δt. We analyze their impact in Section B.1. The detailed settings can be found in Table 4.

B Additional Experiment Results

B.1 Analysis of Hyperparameters

Effect of step Δt. The hyperparameter Δt determines the number of adaptation steps the model takes after each pruning-expansion operation. As shown in Figure 6, when Δt is too small (e.g., 200), the model cannot quickly adapt to the pruning and expansion due to the excessive number of operations, resulting in suboptimal performance. As Δt increases, performance gradually improves, reaching its peak at $n = 1200$ with five pruning-expansion operations. When Δt continues to increase, the number of operations becomes too few, diminishing the benefits of the pruning-expansion

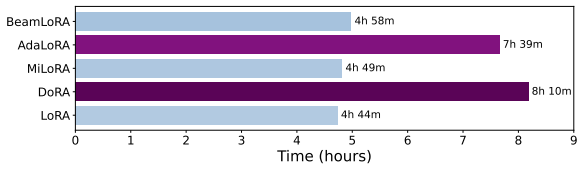


Figure 8: Fine-tuning time for different methods on LLaMA2-7B with $r = 64$.

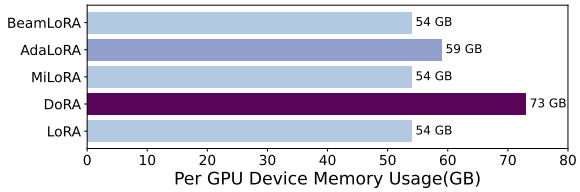


Figure 9: Fine-tuning memory usage for different methods on LLaMA2-7B with $r = 64$.

operation. In our implementation, we determine the number of BeamLoRA operations based on the total number of fine-tuning steps. Specifically, for the MetaMathQA dataset, we select from {2500, 3000, 3500}; for the CodeFeedback dataset, we select from {800, 1000, 1200}; and for the Commonsense dataset, we select from {2400, 2800, 3200} for Δt.

Effect of p_{init}. The hyperparameter p_{init} determines the number of affected ranks in each pruning-expansion operation. According to Eq. 14, a smaller p_{init} results in more ranks being operated. As shown in Figure 7, we see that when a larger number of ranks is operated (e.g., $p = 0.93$), the performance, while still better than LoRA, is not optimal. As p_{init} increases, the number of operated ranks decreases, reaching optimal performance at $p = 0.95$. However, if p_{init} continues to increase, the number of operated ranks further decreases, reducing the advantages of the pruning-expansion process, and performance gradually declines to levels comparable to LoRA.

B.2 Fine-tuning Efficiency

The efficiency of fine-tuning primarily involves two aspects: fine-tuning time and memory usage. We present the fine-tuning time for different methods on the MetaMathQA dataset in Figure 8. We see that BeamLoRA and MiLoRA require a similar time as LoRA. However, DoRA and AdaLoRA require 1.73 times and 1.62 times the fine-tuning time of LoRA, respectively, thereby losing some of the time-saving advantages that LoRA offers.

In terms of memory usage, as shown in Figure 9,

	Math Reasoning		Code Generation		Commonsense Reasoning	
Base Model	LLaMA2-7B	Mistral-7B	LLaMA2-7B	Mistral-7B	LLaMA2-7B	LLaMA2-13B
Rank r	64	64	64	64	32	32
p_{init}	0.95	0.95	0.95	0.95	0.96	0.96
Step Δt	3000	2500	1200	800	3200	2400

Table 4: Detailed settings of hyperparameters in BeamLoRA. In order to thoroughly evaluate the method's performance, we utilize two standard configurations in our experiments: $r = 64$ and $r = 32$. The p_{init} is based on the rank size. Δt is determined based on the total number of fine-tuning steps across different datasets to control the number of operations.

it can be observed that BeamLoRA and MiLoRA also have similar memory usage on each GPU as LoRA. AdaLoRA requires slightly more memory compared to these three, while DoRA requires 1.35 times the memory of LoRA. Overall, in terms of fine-tuning time and memory usage, BeamLoRA is similar to LoRA and significantly lower than DoRA and AdaLoRA, rendering it practical.