DropLoRA: Sparse Low-Rank Adaptation for Parameter-Efficient Fine-Tuning

Anonymous ACL submission

Abstract

LoRA-based large model parameter-efficient fine-tuning (PEFT) methods use low-rank decomposition to approximate updates to model parameters. However, compared to full-parameter fine-tuning, low-rank updates often lead to a performance gap in downstream tasks. To address this, we introduce DropLoRA, a novel pruning-based approach that focuses on pruning the rank dimension. Unlike conventional methods that attempt to overcome the low-rank bottleneck, DropLoRA innovatively integrates a pruning module between the two low-rank matrices in LoRA to simulate dynamic subspace learning. This dynamic low-rank subspace learning allows DropLoRA to overcome the limitations of traditional LoRA, which operates within a static subspace. By continuously adapting the learning subspace, DropLoRA significantly boosts performance without incurring additional training or inference costs. Our experimental results demonstrate that DropLoRA consistently outperforms LoRA in fine-tuning the LLaMA series across a wide range of large language model generation tasks, including commonsense reasoning, mathematical reasoning, code generation, and instruction-following. Our code will be available after the anonymous review.

1 Introduction

Large language models (LLMs) have demonstrated remarkable proficiency in diverse cognitive tasks spanning machine translation, information extraction, question answering, human-like dialogue systems, and logical reasoning (Guo et al., 2025; Achiam et al., 2023; Brown et al., 2020). While this methodology typically involves two-phase training - pre-training on extensive datasets followed by instruction-based fine-tuning (IFT) for downstream task optimization. The substantial computation and memory overhead required for effective instruction fine-tuning pose significant barriers to implementing these architectures in resource-constrained scenarios (Grattafiori et al., 2024). Consequently, Efficient fine-tuning techniques based on models are increasingly gaining popularity and attention within the community.

Parameter-efficient fine-tuning (PEFT) methods aim to achieve performance comparable to full-parameter fine-tuning by freezing the majority of the large model's parameters and fine-tuning a small number of parameters on downstream tasks (Ding et al., 2023). Based on this fundamental idea, Low-Rank Adaptation (LoRA) technology approximates model parameter updates by introducing two low-rank matrices, and it has garnered widespread attention within the community in recent years (Hu et al., 2022). Mathematically, the original model weight W can be reparametered into $W = W_0 + BA$, where $W \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{m\times r}$, $A \in \mathbb{R}^{r\times n}$. Because of the rank $r \ll \min\{m, n\}$, the learnable parameters are far smaller than the original weight parameter count, which saves much GPU memory. Despite LoRA's high flexibility and broad applicability, its performance is constrained by the rank r of the low-rank matrices A and B, resulting in it still slightly underperforming compared to full-parameter fine-tuning (Xia et al., 2024).

To address the performance limitations of LoRA, the research community has investigated a diverse array of strategies. A number of works have focused on the initialization of LoRA, employing singular value decomposition (SVD) of the original matrices to optimize the initialization of the low-rank matrices A and B (Meng et al., 2024a; Wang et al., 2025; Lingam et al., 2024; Büyükakyüz, 2024). Another line of research aims to enhance the rank of LoRA by refining the low-rank matrices to mitigate the rank bottleneck and improve its expressive power (Meng et al., 2024b; Wang et al., 2025; Jiang et al., 2024). Additionally, some techniques

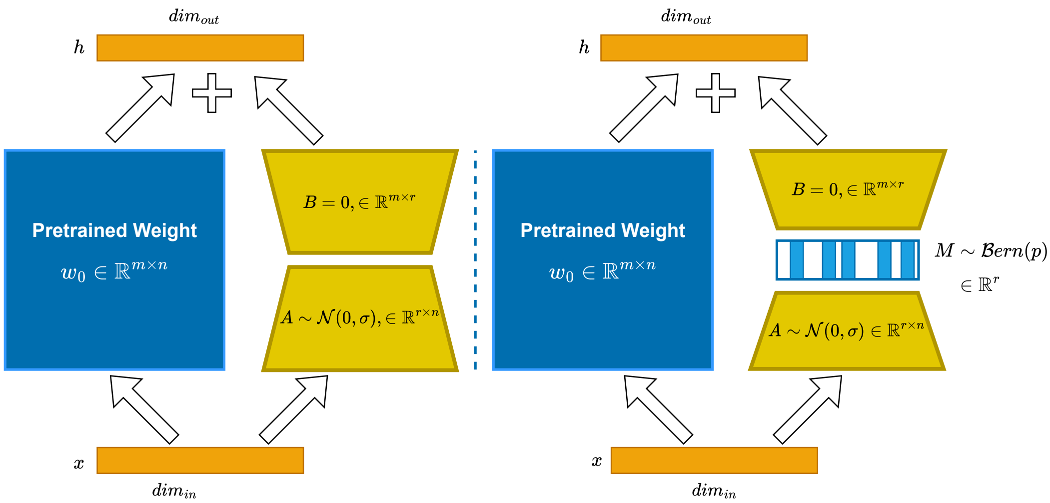

Figure 1: Schematic comparison of LoRA (left) and DropLoRA (right). In LoRA, the original weights remain unchanged, with updates being applied solely to two low-rank matrices. DropLoRA introduces a mask matrix M between these two low-rank matrices to enable pruning. At each parameter iteration step, a distinct M is sampled from a Bernoulli distribution, enabling subspace learning. In both scenarios, the low-rank matrices and vectors can be seamlessly integrated into the original weight matrix W, thereby introducing no additional latency.

dynamically adapt the rank of LoRA for different weights, offering greater flexibility and efficiency (Valipour et al., 2022; Zhang et al., 2023).

In contrast to reparameterizing the original weights, Zhao et al. (2024) introduces GaLore, an innovative approach that achieves memory efficiency by projecting gradients onto diverse low-rank subspaces—effectively reparameterizing the gradients—while demonstrating exceptional performance. While Galore utilizes gradient-based low-rank projection, it fundamentally differs from LoRA, representing a distinct methodology. A key distinction is that LoRA fine-tunes only a subset of parameters, whereas GaLore optimizes all parameters. Inspired by the efficiency of the dynamic subspace learning of GaLore, we are prompted to investigate: Can LoRA further enhance its performance through the application of dynamic subspace learning?

Building on this insight, we propose DropLoRA, a strategy that simulates subspace learning by dynamically adjusting the rank of LoRA. For a fixed rank, LoRA operates within a consistent low-rank subspace, with the learning subspace remaining static throughout the process. To simulate dynamic subspace learning, we propose a simple yet effective pruning strategy, as illustrated in Figure 1. By applying unified dynamic pruning to the two low-rank matrices, each pruning operation corresponds to a distinct subspace. Specifically, we sample the rank-dimension pruning matrix M from a Bernoulli distribution, hence, $M \in \{0,1\}^r, M_i \overset{\text{i.i.d.}}{\sim} \text{Bernoulli}(p), i \in \{1, 2, ..., r\}$, where p is the pruning probability, r is the rank of A and B. Hence our DropLoRA can be formulated as $W = W_0 + (B \odot M) \times (M \odot A)$. The dynamics of subspace learning are reflected in sampling different pruning matrices M at each iteration step.

Extensive experiments show that subspace learning, as exemplified by DropLoRA, can serve as a novel optimization direction. In summary, our main contributions are as follows:

- We propose DropLoRA, an innovative optimization strategy for LoRA, which for the first time introduces subspace learning into the LoRA framework, exploring a novel direction for optimization in the community.
- Our pruning strategy is designed to be seamlessly integrated into any LoRA variant without introducing additional computational or storage overhead, showcasing its adaptability and practicality.
- DropLoRA achieves state-of-the-art (SOTA) performance across diverse domains, highlighting its broad applicability and robustness.

2 Related Work

Parameter-Efficient Fine-Tuning (PEFT) methods for supervised fine-tuning of large models have become increasingly significant, particularly in resource-constrained scenarios. The development of various efficient fine-tuning methods has emerged as a prominent research focus. Existing efficient fine-tuning techniques can be categorized into the following aspects. Methods based on adapters aim to insert different adapter layers between the layers of a model for various downstream tasks (Houlsby et al., 2019; He et al., 2021; Mahabadi et al., 2021).

Prompt-based methods, such as P-tuning (Liu et al., 2021) and prefix-tuning (Li and Liang, 2021) , introduce continuous prompt tokens into the input space, allowing for efficient adaptation of large PLMs by only fine-tuning these learned prompt embeddings while keeping the original model parameters frozen. These approaches differ from traditional fine-tuning, as they avoid direct modification of the underlying model weights, instead relying on task-specific soft prompts to guide the model's behavior. However, both adapter-based and prompt-based approaches modify the model's internal structure, either by inserting additional trainable layers or by prepending learnable prompt embeddings. While these methods significantly reduce the number of trainable parameters compared to full fine-tuning, they inevitably introduce additional computational overhead during both training and inference. Specifically, the inclusion of extra parameters or prompt tokens increases memory usage and may lead to higher inference latency, particularly in real-time applications where low-latency responses are critical.

LoRA-based methods and their variants achieve parameter efficiency by decomposing the base weight matrix into two low-rank matrices, demonstrating significant advantages in deployment, particularly for mobile device applications (Hu et al., 2022; Kopiczko et al., 2023; Meng et al., 2024b; Liu et al., 2024; Zhang et al., 2023; Meng et al., 2024a). For diverse applications, it is only necessary to store distinct LoRA adapters specifically fine-tuned for their respective downstream tasks. Variants of LoRA primarily include optimization of initialization parameters, rank enhancement, and adaptive rank selection, among others (Meng et al., 2024a,b; Valipour et al., 2022). Extensive research related to LoRA demonstrates that LoRA-based methods are currently dominating the field of Parameter-Efficient Fine-Tuning (PEFT).

Algorithm 1 DropLoRA, torch-style pseudocode.

```
class DropLoRALayer(nn.Module):
    def __init__(
    self,
    r: int = 32, # rank
    p: float = 0.5, # pruning probability
    d1: int = 4096, # input dimension
    d2: int = 4096, # output dimension
    base_layer: nn.Module # pre-trained layer
    ):
    self.base_layer = base_layer
    self.A = torch.randn(r, d1)
    self.B = torch.zeros(d2, r)
    self.M = Dropout(p) ## Line 1.
    self.base_layer.freeze()

    def forward(self, x: torch.Tensor):
        h = self.base_layer(x)
        ## In LoRA
        ## delta = x @ self.A @ self.B
        ## Line 2
        delta = self.M(x @ self.A) @ self.B

        return h + delta
```

Subspace Learning focuses on deriving low-dimensional, essential features from high-dimensional data to enhance learning efficiency and effectiveness. By eliminating redundancy and capturing essential characteristics, subspace learning facilitates more efficient and effective learning processes, enhancing both computational performance and model accuracy (Liu et al., 2012; De La Torre and Black, 2003). Extensive research has demonstrated that subspace learning exhibits excellent generalization capabilities, making it a robust approach for various machine learning tasks (Hinton and Salakhutdinov, 2006; Wright et al., 2008; Zhao et al., 2024). LoRA assumes that weight updates occur in a low-rank space; however, due to its static rank nature, it can essentially be regarded as a form of static subspace learning.

3 Method

LoRA reparameterizes the update of the original weights as the product of two low-rank matrices, as expressed in Equation 1:

$$h = W_0 x + \underline{BA}x \tag{1}$$

where $W_0 \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{m \times r}$, $A \in \mathbb{R}^{r \times n}$. During the training process, the original weights W_0 remain unchanged, with updates being applied exclusively to the weights of the two low-rank matrices A, B. Here, We undeline the parameters updated during the training process. Because of the rank $r \ll \min\{m, n\}$, the learnable parameters are far smaller than the original weight parameter count. When the rank is fixed, LoRA can be viewed as learning within a static subspace, which may inherently constrain its expressive capacity.

To simulate dynamic subspace learning, we propose a pruning technique based on dynamic masking. Specifically, we sample the rank dimension using a Bernoulli distribution $\mathrm{Bernoulli}\,(p)$ to generate a mask vector of rank size, where a value of 1 retains the dimension and a value of 0 discards it. Our DropLoRA method is expressed as:

$$h = W_0 x + (\underline{B} \odot M)\,(M \odot \underline{A})\,x \qquad (2)$$

where $M \in \{0,1\}^r, M_i \overset{\text{i.i.d.}}{\sim} \mathrm{Bernoulli}\,(p)\,, i \in \{1, 2, ..., r\}$ and $\odot$ represents element-wise multiplication. M is a mask matrix, where p represents the pruning probability. At each training iteration step, we randomly sample a distinct mask matrix M to simulate dynamic subspace learning.

Rank Analysis When we apply the sampled mask to prune the rank dimension, it implies that the effective rank of the two low-rank matrices $\tilde{A} = (M \odot A)$ and $\tilde{B} = (B \odot M)$ is reduced compared to their original rank. With a pruning probability of 0.5, the rank of $\tilde{A}$ and $\tilde{B}$ becomes only half of the original rank.

Equivalence Intuitively, there are two pruning strategies: one applies a unified pruning using the same mask matrix for both low-rank matrices, while the other prunes A and B separately using distinct mask matrices. For the latter, the two mask matrices operate under a logical AND relationship, effectively equivalent to their intersection. This is functionally identical to using a single mask matrix with values equal to the intersection. Thus, the two approaches are equivalent.

Easy Implementation Since the product of LoRA's two low-rank matrices is mathematically equivalent to a two-layer perceptron without activation, the masked pruning strategy effectively functions as dropout applied to the intermediate hidden layer. This implies that, compared to LoRA, DropLoRA can be implemented with just two additional lines of code, as illustrated in Algorithm 1.

It is worth noting that, although similar in implementation, our method differs from traditional Dropout regularization methods (Srivastava et al., 2014). The Dropout method generally randomly drops some of the high-dimensional inputs. In contrast, our method randomly discards the rank dimension of LoRA low-rank matrices. Intuitively, the expressive power of LoRA is limited by the size of the rank. Randomly discarding the rank dimension will further reduce the expressive power of LoRA, resulting in severe performance degradation. Therefore, pruning the rank dimension is somewhat counterintuitive. However, dynamic low-rank subspace learning allows DropLoRA to overcome the limitations of traditional LoRA, which operates within a static subspace and the model is prompted to learn more intrinsic parameter variation characteristics, thereby improving performance.

Training and Inference During the training process, at each iteration step, we obtain different low-rank subspaces by sampling different pruning vectors through the Bernoulli distribution. During backpropagation, only the retained parameters are involved in the update. In the reasoning process, in order to enhance the model's expressive power, we do not use the pruning module. By integrating the parameters learned in different subspaces, this has a similar effect to ensemble learning, thereby improving the robustness of the model.

4 Experiments

To evaluate the effectiveness of the DropLoRA method, we conducted extensive experiments encompassing commonsense reasoning tasks, mathematical tasks, coding tasks, instruction following tasks. For all tasks, we choose the same LoRA-related baselines including:

LoRA(Hu et al., 2022) decomposes a parameter update into the product of two low-rank matrices, where one matrix is initialized with Gaussian distribution and the other is initialized with zeros.

DoRA(Liu et al., 2024) decouples the magnitude and direction of the parameter update, using LoRA to update the direction and a learnable magnitude vector to update the magnitude.

PiSSA(Meng et al., 2024a) initializes LoRA by applying singular value decomposition (SVD) to the pre-trained weights, using the **Principal Singular Components** to initialize LoRA, while the residual components are used to initialize the pre-trained weights.

MiLoRA(Wang et al., 2025) initializes LoRA by applying singular value decomposition (SVD) to the pre-trained weights, using the **Minor Singular**

Model	PEFT	# Parameters	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
ChatGPT†	–	–	73.1	85.4	68.5	78.5	66.1	89.8	79.9	74.8	77.0
LLaMA2-7B	LoRA†	56.10M	69.8	79.9	79.5	83.6	82.6	79.8	64.7	81.0	77.6
	DoRA†	56.98M	71.8	83.7	76.0	89.1	82.6	83.7	68.2	82.4	79.7
	PiSSA⋆	56.10M	67.6	78.1	78.4	76.6	78.0	75.8	60.2	75.6	73.8
	MiLoRA†	56.10M	67.6	83.8	80.1	88.2	82.0	82.8	68.8	80.6	79.2
	LoRA	56.10M	74.37	**87.38**	**81.32**	95.07	85.79	88.80	75.34	<u>87.00</u>	84.38
	DoRA	56.98M	<u>74.46</u>	86.18	80.60	94.91	<u>87.53</u>	89.14	75.85	86.40	84.39
	PiSSA	56.10M	73.27	82.59	79.84	92.88	84.77	85.23	71.33	84.80	81.84
	MiLoRA	56.10M	**74.53**	86.45	80.81	<u>95.23</u>	86.90	<u>89.48</u>	<u>76.54</u>	86.00	<u>84.49</u>
	DropLoRA (Ours)	56.10M	74.22	<u>87.00</u>	<u>80.91</u>	**95.24**	**87.61**	**89.73**	**77.30**	**87.20**	**84.91**
LLaMA3-8B	LoRA†	56.62M	70.8	85.2	79.9	91.7	84.3	84.2	71.2	79.0	80.8
	DoRA†	57.41M	74.6	89.3	79.9	95.5	85.6	90.5	80.4	85.8	85.2
	PiSSA⋆	56.62M	67.1	81.1	77.2	83.6	78.9	77.7	63.2	74.6	75.4
	MiLoRA†	56.62M	68.8	86.7	77.2	92.9	85.6	89.48	75.5	81.8	81.9
	LoRA	56.62M	75.54	89.06	81.12	95.99	88.08	92.80	82.59	89.20	86.78
	DoRA	57.41M	75.41	89.12	881.27	95.83	87.69	92.42	82.59	89.00	86.67
	PiSSA	56.62M	72.66	86.13	80.14	93.60	84.77	89.73	76.88	86.00	83.74
	MiLoRA	56.62M	74.53	86.45	80.81	95.23	86.90	89.48	76.54	86.00	84.49
	DropLoRA (Ours)	56.62M	**76.45**	**90.04**	**82.19**	**96.59**	**89.34**	**93.18**	**83.28**	**89.80**	**87.61**

Table 1: Commonsense reasoning evaluation results for LLaMA2-7B and LLaMA3-8B on eight tasks. The reported metric in this table is accuracy. †Results are cited from the original paper and ⋆results are cited from Wang et al. (2025). For PEFT results, all other experiments without superscripts are performed by ourselves. Bold numbers indicate the highest performance scores and underline numbers indicate the second performance scores for each dataset across the different PEFT methods for the corresponding model.

Components to initialize LoRA, while the residual components are used to initialize the pre-trained weights.
All experiments are conducted on $4 \times A100$ GPUs with Deepspeed ZERO-2 stage(Rasley et al., 2020) to accelerate training.

4.1 Commensense Reasoning

To evaluate the impact of efficient fine-tuning techniques on commonsense knowledge and logical reasoning abilities, we conduct experiments on commonsense knowledge reasoning tasks.

Datasets The commonsense reasoning dataset consists of 8 sub-tasks, each with its own predefined training and testing sets, including BoolQ(Clark et al., 2019), PIQA(Bisk et al., 2020), SIQA(Sap et al., 2019), HellaSwag(Zellers et al., 2019), WinoGrande(Sakaguchi et al., 2021), ARC-e, ARC-c(Clark et al., 2018) and OBQA(Mihaylov et al., 2018). We follow the experimental setup from Hu et al. (2023), where the training sets of the 8 sub-tasks are combined, and inference and evaluation are conducted separately on their respective testing sets.

Experimental Setting We choose LLaMA2-7B(Touvron et al., 2023)[1] and LLaMA3-8B(Grattafiori et al., 2024)[2] as our backbone models. To ensure a fair comparison, we implement all PEFT experiments ourselves. We also report chatGPT-api based results sourced from Liu et al. (2024). For the hyperparameter configuration, we also follow the parameter settings from Hu et al. (2023). Note that, to accelerate training, we use a batch size of 128 instead of the original configuration of 16. For all other hyperparameters, we strictly follow the parameter settings from Hu et al. (2023). It is important to note that in all experiments, for DropLoRA, we only adjust the pruning probability and do not adjust any other hyperparameters. For detailed hyperparameter configurations, see Appendix A.

Result Table 1 presents the experimental results for the common-sense reasoning task. We also report the evaluation results based on the ChatGPT API as outlined in the DoRA paper (Liu et al., 2024), which are obtained with the GPT-3.5-turbo API using a zero-shot Chain of Thought approach.

As can be seen, on the LLaMA2-7B, DropLoRA achieved the best performance on five datasets (HellaSwag, WinoGrande, ARC-e, ARC-c, OBQA), the second-best performance on two datasets (PIQA, SIQA), and the best average performance across all eight datasets with an average performance in-

[1] https://hf-mirror.com/meta-llama/Llama-2-7b-hf

[2] https://hf-mirror.com/meta-llama/Meta-Llama-3-8B

Model	Method	# Parameters	GSM8K	MATH	Average
LLaMA2-7B	Full FT†	6738M	66.5	19.8	43.2
	LoRA†	112.20M	60.6	16.9	38.7
	PiSSA†	112.20M	58.2	15.8	37.0
	MiLoRA†	112.20M	63.5	17.8	40.7
	LoRA	112.20M	65.66	16.02	40.84
	DoRA	113.07M	66.19	16.14	41.16
	PiSSA	112.20M	64.37	15.96	40.16
	MiLoRA	112.20M	64.52	14.92	39.72
	DropLoRA (Ours)	112.20M	**66.72**	**16.38**	**41.55**
LLaMA3-8B	LoRA	113.25M	80.44	30.46	55.45
	DoRA	114.03M	80.44	30.21	55.32
	PiSSA	113.25M	79.53	28.92	54.22
	MiLoRA	113.25M	80.74	30.62	55.68
	DropLoRA (Ours)	113.25M	**81.32**	**30.74**	**56.03**

Table 2: Math reasoning evaluation results for GSM8K and MATH based on LLaMA2-7B and LLaMA3-8B. †Results are cited from Wang et al. (2025) and All other experiments without superscripts are performed by ourselves.

Model	Method	# Parameters	HumanEval	MBPP	Average
LLaMA2-7B	Full FT†	6738M	21.34	35.59	28.47
LLaMA2-7B	LoRA	56.10M	18.90	41.27	30.09
	DoRA	56.98M	14.63	42.86	28.75
	PiSSA	56.10M	9.76	42.86	26.31
	MiLoRA	56.10M	**21.34**	37.57	29.46
	DropLoRA (Ours)	56.10M	**21.34**	**43.39**	**32.37**
LLaMA3-8B	LoRA	56.62M	**62.81**	65.87	64.34
	DoRA	57.41M	59.76	66.40	63.08
	PiSSA	56.62M	57.93	65.87	61.90
	MiLoRA	56.62M	59.76	66.93	63.35
	DropLoRA (Ours)	56.62M	60.37	**70.37**	**65.37**

Table 3: Code evaluation results for HumanEval and MBPP based on LLaMA2-7B and LLaMA3-8B. †Results are cited from Meng et al. (2024a) and the other experimental results are from ourselves.

crease of $+0.53$ points compared to LoRA. On the LLaMA3-8B model, DropLoRA achieves the best performance on all eight datasets, with an average performance increase of $+0.83$ points compared to LoRA, indicating that DropLoRA is an effective parameter-efficient fine-tuning method. We observe that both LoRA and DoRA achieve comparable performance on LLaMA2-7B and LLaMA3-8B. MiLoRA slightly outperforms LoRA on LLaMA2-7B, but shows a significant performance gap on LLaMA3-8B. PiSSA, on the other hand, performs substantially worse than other methods on both models. This indicates instability in performance for methods that fine-tune either the principal or the minor singular components. In contrast, our method achieves the best performance on both models, demonstrating its superior stability.

4.2 Math and Code Reasoning

To evaluate numerical computation and logical reasoning capabilities, we conduct performance assessments on mathematical problem-solving and programming tasks.

Datasets We evaluate mathematical problem-solving capabilities on MetaMathQA dataset(Yu et al., 2023), including 395K samples generated by augmenting the training sets of GSM8K(Cobbe et al., 2021) and MATH(Hendrycks et al., 2021). During the testing phase, we perform inference and evaluate performance on the test sets of GSM8K and MATH separately.

To evaluate code capabilities, we fine-tune on the CodeFeedback(Zheng et al., 2024) dataset and perform evaluation on the HumanEval(Chen et al., 2021)and MBPP(Austin et al., 2021) test sets.

Experimental Setting We choose LLaMA2-7B[1] and LLaMA3-8B[2] as our pre-trained models. We use hyperparameter configurations similar to those for commonsense reasoning. For the mathematical reasoning task, due to the large training set of MetaMathQA, we set the rank of LoRA to 64 and train for only one epoch to avoid overfitting. For the code evaluation task, we maintain the same hyperparameter configuration as for commonsense reasoning. For detailed hyperparameter configurations, see Appendix A.

Result Tables 2 and Table 3 present the experimental results for mathematical reasoning and code reasoning tasks, respectively. DropLoRA consistently achieves state-of-the-art performance across all four reasoning tasks, demonstrating its effectiveness in handling both mathematical and code-based problem-solving scenarios.

Notably, on mathematical reasoning tasks with LLaMA2-7B, DropLoRA outperforms standard LoRA by an average margin of $+0.7$ percentage points, while this advantage expands to $+2.28$ percentage points on coding tasks. The performance gap persists with LLaMA3-8B, where DropLoRA achieves $+0.58$ and $+1.03$ percentage point improvements over LoRA in mathematical and coding tasks, respectively.

We also observed that LoRA and DoRA achieved comparable performance on mathematical reasoning tasks, while MiLoRA exhibited significant performance fluctuations across two models. On coding tasks, DoRA, MiLoRA, and PiSSA all show substantial performance gaps compared to LoRA, hinting at the complexity of coding tasks. Despite this, our method still significantly outperformed LoRA. Specifically, on LLaMA2-7B, it surpassed LoRA by $+2.3$ percentage points; on LLaMA3-

Model	Method	# Parameters	MT-Bench
LLaMA2-7B	LoRA	56.10M	5.16
	DoRA	56.98M	5.33
	PiSSA	56.10M	5.20
	MiLoRA	56.10M	5.25
	DropLoRA (Ours)	56.10M	**5.54**
LLaMA3-8B	LoRA	56.62M	6.31
	DoRA	57.41M	6.09
	PiSSA	56.62M	5.94
	MiLoRA	56.62M	6.11
	DropLoRA (Ours)	56.62M	**6.42**

Table 4: Instruction following results based on LLaMA2-7B and LLaMA3-8B, assigned by GPT-4 to the answers. All experimental results are conducted by ourselves.

8B, it surpassed LoRA by +1 point. The absolute leading advantage demonstrates the superior performance of our method on reasoning tasks.

4.3 LLM Capability for Open Questions

To comprehensively evaluate our model's capacity for handling open-ended questions and executing complex instructions, we employ the MT-Bench dataset (Zheng et al., 2023), a widely recognized benchmark in the field of natural language processing. This meticulously curated dataset contains 80 carefully designed questions spanning diverse domains and difficulty levels, along with 3,300 expert-annotated pairwise human preference judgments comparing responses generated by six different models.

Experimental Setting We utilize the same hyperparameters as those used in the Hu et al. (2023). For the evaluation of conversational abilities, we employ the method mentioned in the MT-Bench paper(Zheng et al., 2023)[3], utilizing GPT-4 to score the dialogue tasks. For detailed hyperparameter configurations, see Appendix A.

Result Table 4 displays the results of our experiments conducted on the dialogue task. Our proposed method demonstrates the best performance across both models. Specifically, when compared to LoRA, the performance improves by +0.38 points on LLaMA2-7B and by +0.38 points on LLaMA3-8B. Notably, we observe that both PiSSA and MiLoRA, in comparison to LoRA, yield nearly the same marginal gains in the dialogue task. This suggests that the differences in performance between fine-tuning the principal singular component (PiSSA) and fine-tuning the minor singular component (MiLoRA) are relatively minor and do not have a significant impact on this particular task.

[3] https://github.com/lm-sys/fastchat

Method	rank	pruning rate	accuracy
LoRA	16	0	84.5
LoRA	32	0	84.4
DropLoRA (Ours)	32	0.5	**84.9**

Table 5: The performance comparison of LoRA and DropLoRA on inference tasks with different ranks and pruning rates.

4.4 Study

Effect of Pruning Module DropLoRA inserts a pruning module between the two low-rank matrices of LoRA to simulate subspace learning, while keeping everything else consistent with LoRA. As can be seen from Table 1 $\sim$ Table 4, on all four tasks, whether it is on the LLaMA2-7B or LLaMA3-8B model, the performance of DropLoRA is significantly better than that of LoRA, proving the effectiveness of the pruning module and its generalization to different tasks and models. As shown in Table 5, for DropLoRA, when the rank is 32 and the pruning rate is 0.5, it means that only half of the parameters are updated during each parameter update, which is comparable to the LoRA parameters with rank 16. However, regardless of whether the rank of LoRA is 16 or 32, DropLoRA consistently outperforms LoRA, proving the effectiveness of the DropLoRA pruning module.

Effect of Pruning Rate We explore the impact of different pruning rates on experimental results, such as the pruning rate in Equation 2. Figure 2 shows the fine-tuning performance of different pruning rates on the commonsense reasoning, math and coding tasks. We can see that when the pruning rate varies within the range of $0.1 \sim 0.5$, the performance fluctuates slightly. When the pruning rate is 0.3, compared with LoRA, the performance improvement is the greatest. We observe that when the pruning rate is set to 0.5, it starts to perform worse than LoRA on math and code tasks. This is because when the pruning rate is too large, it will reduce the low-rank subspace representation ability of the model, resulting in performance degradation. We also observe that even when the pruning rate is set to 0.5, which means that only half of the parameters are activated during the training process of each subspace, DropLoRA can still achieve better performance than LoRA on the commonsense reasoning task. This demonstrates the effectiveness

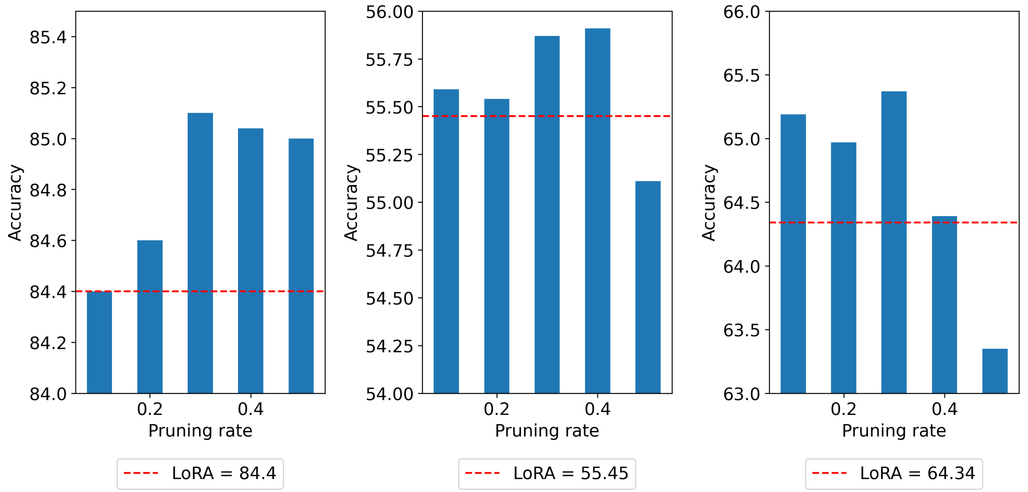

Figure 2: Average accuracy of LoRA and DropLoRA for varying pruning rate on the commonsense reasoning, math and code tasks. The left, middle, and right figures correspond to commonsense reasoning, math, and coding tasks respectively.

of subspace learning.

Parameter Scalability We conduct an exploration into the relationship that exists between the quantity of trainable parameters and the performance of both the Low-Rank Adaptation (LoRA) method and our proposed method. We set the rank $r = \{8, 16, 32, 64\}$, and α remains twice the rank. Other hyperparameters remain consistent with those of the commensense reasoning task. The average accuracy of LoRA and DropLoRA for varying ranks for LLaMA-7B on the commonsense reasoning tasks is depicted in Figure 3. As shown in Figure 3, under all rank configurations, DropLoRA consistently outperforms LoRA. Due to the structural similarity between the two, their performance trends are also similar. When the rank is larger, DropLoRA's performance remains significantly superior to that of LoRA. However, when the rank is smaller, the performance gap between the two narrows. This is because, when the rank is small, DropLoRA, due to the pruning module, learns in a lower-rank subspace compared to LoRA. An excessively low rank can limit the expressive power of the subspace learning.

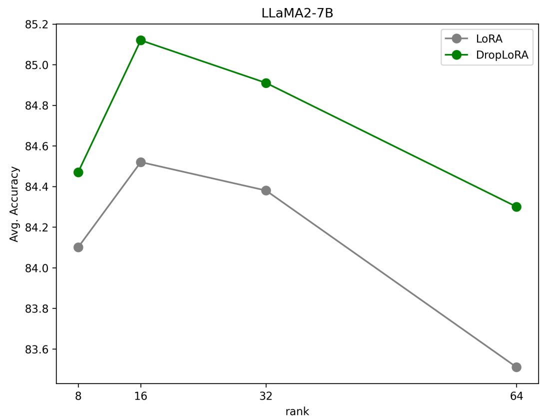

Figure 3: Average accuracy of LoRA and DropLoRA for varying ranks for LLaMA-7B on the commonsense reasoning tasks.

5 Conclusion

In this paper, we introduce DropLoRA, a simple yet effective low-rank adaptive method for parameter-efficient fine-tuning of large language models. By inserting a pruning module between the two low-rank matrices of LoRA to simulate subspace learning, we show that performance can be improved not only by increasing LoRA's rank but also by lowering it. We validate the effectiveness of DropLoRA on a wide range of large language model evaluation benchmarks, including commonsense reasoning, math reasoning, code generation, and instruction-following tasks. Experimental results indicate that DropLoRA consistently outperforms other baseline methods, including LoRA, DoRA, PiSSA, and MiLoRA, across all tasks. Compared to LoRA, DropLoRA introduces no additional parameters, thus not increasing any training or inference costs. Our research shows that, in addition to increasing the rank of LoRA, lowering its rank can also enhance the performance, providing a new perspective for future optimization on parameter-efficient fine-tuning of LLMs.

Limitations

Due to computational resource constraints, we have only validated the effectiveness of DropLoRA on large model generation tasks, such as commonsense reasoning, math reasoning, code generation, and instruction-following tasks. However, an interesting future direction is whether DropLoRA can enhance performance on multimodal large model benchmark tasks beyond language generation. Another open question is whether we can provide a theoretical foundation to support the effectiveness of rank reduction for simulating subspace learning. We consider these unresolved issues as important areas for future research.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Kerim Büyükakyüz. 2024. Olora: Orthonormal low-rank adaptation of large language models. *arXiv preprint arXiv:2406.01775*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

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. *arXiv preprint arXiv:1905.10044*.

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

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

Fernando De La Torre and Michael J Black. 2003. A framework for robust subspace learning. *International Journal of Computer Vision*, 54:117–142.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. 2023. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

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

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. *arXiv preprint arXiv:2103.03874*.

Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

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

Ting Jiang, Shaohan Huang, Shengyue Luo, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, Qi Zhang, Deqing Wang, and Fuzhen Zhuang. 2024. Mora: High-rank updating for parameter-efficient fine-tuning. *Preprint*, arXiv:2405.12130.

Dawid J Kopiczko, Tijmen Blankevoort, and Yuki M Asano. 2023. Vera: Vector-based random matrix adaptation. *arXiv preprint arXiv:2310.11454*.

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

Vijay Lingam, Atula Tejaswi, Aditya Vavre, Aneesh Shetty, Gautham Krishna Gudur, Joydeep Ghosh, Alex Dimakis, Eunsol Choi, Aleksandar Bojchevski, and Sujay Sanghavi. 2024. Svft: Parameter-efficient fine-tuning with singular vectors. *Preprint*, arXiv:2405.19597.

Guangcan Liu, Zhouchen Lin, Shuicheng Yan, Ju Sun, Yong Yu, and Yi Ma. 2012. Robust recovery of subspace structures by low-rank representation. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):171–184.

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 *Forty-first International Conference on Machine Learning*.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2021. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. 2021. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. *arXiv preprint arXiv:2106.04489*.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. 2024a. Pissa: Principal singular values and singular vectors adaptation of large language models. *Preprint*, arXiv:2404.02948.

Xiangdi Meng, Damai Dai, Weiyao Luo, Zhe Yang, Shaoxiang Wu, Xiaochen Wang, Peiyi Wang, Qingxiu Dong, Liang Chen, and Zhifang Sui. 2024b. Periodiclora: Breaking the low-rank bottleneck in lora optimization. *arXiv preprint arXiv:2402.16141*.

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. *arXiv preprint arXiv:1809.02789*.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3505–3506.

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

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

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. 2022. Dylora: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. *arXiv preprint arXiv:2210.07558*.

Hanqing Wang, Yixia Li, Shuo Wang, Guanhua Chen, and Yun Chen. 2025. Milora: Harnessing minor singular components for parameter-efficient llm finetuning. *Preprint*, arXiv:2406.09044.

John Wright, Allen Y Yang, Arvind Ganesh, S Shankar Sastry, and Yi Ma. 2008. Robust face recognition via sparse representation. *IEEE transactions on pattern analysis and machine intelligence*, 31(2):210–227.

Wenhan Xia, Chengwei Qin, and Elad Hazan. 2024. Chain of lora: Efficient fine-tuning of language models via residual learning. *arXiv preprint arXiv:2401.04151*.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*.

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

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. 2024. Galore: Memory-efficient llm training by gradient low-rank projection. *Preprint*, arXiv:2403.03507.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue.

2024. Opencodeinterpreter: Integrating code generation with execution and refinement. *arXiv preprint arXiv:2402.14658*.

A Appendix

Table 6 presents the statistics of the datasets used in this paper.

A.1 Dataset Statistics

Dataset	Domain	# train	# test	Answer
BoolQ	CS	9.4K	3,270	Yes/No
PIQA	CS	16.1K	1,830	Option
SIQA	CS	33.4K	1,954	Option
HellaSwag	CS	39.9K	10,042	Option
WinoGrande	CS	63.2K	1,267	Option
ARC-e	CS	1.1K	2,376	Option
ARC-c	CS	2.3K	1,172	Option
OBQA	CS	5.0K	500	Option
GSM8K	Math	240K	1,319	Number
MATH	Math	155K	5,000	Number
Python	Code	104,848	563	Code
Instruction Following	Conversation	143K	80	Text

Table 6: Details of datasets used in our experiment setting including commonsense reasoning, math reasoning, code reasoning and instruction following tasks.

A.2 Our Hyperparameter Setup for LLM

Table 7 presents the hyperparameter configurations used in our experiments. To ensure fairness, our hyperparameter settings are consistent with those reported in the DoRA(Liu et al., 2024) and MiLoRA(Wang et al., 2025) papers. Note that, to accelerate training, the batch size for all experiments in this paper is set to 128.

Hyperparameters	Commonsense	Math	Code	Conversation
Rank r	32	64	32	32
α of LoRA	64	128	64	64
α of DoRA	64	128	64	64
α of DropLoRA	64	128	64	64
α of PiSSA	32	64	32	32
α of MiLoRA	32	64	32	32
Dropout	0.05			
Pruning Rate	$0.1 \sim 0.5$			
Optimizer	AdamW			
LR	3e-4			
LR Scheduler	Linear			
Batch size	128			
Warmup Steps	100			
Epochs	3	1	3	3
Reparameterization	Q,K,V,Up,Down			

Table 7: Our hyperparameter configuration for LLM generation benchmarks for fine-tuning LLaMA2-7B, LLaMA3-8B on the commonsense reasoning, math reasoning, code reasoning and instruction following tasks.

A.3 Case Study

To provide an intuitive demonstration of the effects, we randomly sampled two cases from the mathematical reasoning tasks and presented the reasoning analysis results. Table 8 and Table 9 show the inference results of various methods. In Table 8, except for DoRA, the reasoning processes and outcomes of the other methods are correct. Although DoRA produces the correct result, its reasoning process is incorrect. DropLoRA and LoRA share a similar reasoning process, as do MiLoRA and PiSSA. In Table 9, PiSSA's reasoning process and outcome are both incorrect, while LoRA's reasoning process is correct but its result is wrong. The reasoning processes and results of DropLoRA, DoRA, and MiLoRA are all correct. Note that although the reasoning processes of these three methods are correct, DropLoRA and MiLoRA's reasoning processes explicitly highlight the keyword "least common multiple (LCM)", reflecting a more fundamental reasoning approach.

Question	The ratio of Mary's age to Alice's age is $3:5$. Alice is 30 years old. How many years old is Mary?
DropLoRA	If Alice is 30 years old, then the ratio of Mary's age to Alice's age is $3:5$. This means that for every 3 years old Mary is, Alice is 5 years old. Since Alice is 30 years old, we can set up the equation $\frac{3}{5} \cdot 30 = \boxed{18}$. Therefore, Mary is 18 years old. The answer is: 18.
DoRA	If the ratio of Mary's age to Alice's age is $3:5$, then we can write this as $3x : 5x$, where x is a constant. We are given that Alice is 30 years old, so we can substitute this into the ratio to get $3x : 5x = 30$. To solve for x, we divide both sides of the equation by 3 and 5, respectively, to get $x = 10$. Therefore, Mary's age is $3x = 3(10) = \boxed{30}$ years old. The answer is: 30.
LoRA	If Alice is 30 years old, then the ratio of Mary's age to Alice's age is $3:5$. This means that for every 3 years old Mary is, Alice is 5 years old. Since Alice is 30 years old, we can set up the equation $\frac{3}{5} \cdot 30 =$ Mary's age. Simplifying, we find that Mary is $\frac{3}{5} \cdot 30 = \boxed{18}$ years old. The answer is: 18.
MiLoRA:	If Alice is 30 years old, then the ratio of Mary's age to Alice's age is $3:5$. Let's assume Mary's age is x years old. According to the ratio, we have $\frac{x}{30} = \frac{3}{5}$. Cross-multiplying, we get $5x = 30 \cdot 3$. Simplifying, we find $5x = 90$. Dividing both sides by 5, we get $x = \boxed{18}$. Therefore, Mary is 18 years old. The answer is: 18.
PiSSA	If the ratio of Mary's age to Alice's age is $3:5$, then we can write this as $\frac{\text{Mary's age}}{\text{Alice's age}} = \frac{3}{5}$. We are given that Alice is 30 years old, so we can substitute this into the equation to get $\frac{\text{Mary's age}}{30} = \frac{3}{5}$. To solve for Mary's age, we can cross-multiply to get $5 \cdot \text{Mary's age} = 3 \cdot 30$. Simplifying, we have $5 \cdot \text{Mary's age} = 90$. Dividing both sides by 5, we find that Mary's age = $\boxed{18}$. The answer is: 18.

Table 8: Case Study I for Math Reasoning task on LLaMA2-7B.

Question	What is the smallest number of marbles that could be divided up either into bags of 18 marbles or into bags of 42 marbles, with no marbles left over in each case?
DropLoRA	We can find the least common multiple (LCM) of 18 and 42 to find the smallest number that can be divided evenly into both 18 and 42. Prime factorizing 18 and 42, we have $18 = 2 \times 3^2$ and $42 = 2 \times 3 \times 7$. The LCM is then $2 \times 3^2 \times 7 = 126$. Therefore, the smallest number of marbles that can be divided up into bags of 18 or 42 marbles is $\boxed{126}$. The answer is: 126.
DoRA	We want to find the smallest number that is divisible by both 18 and 42. The prime factorization of 18 is 2×3^2, and the prime factorization of 42 is $2 \times 3 \times 7$. To find the smallest number divisible by both 18 and 42, we take the highest power of each prime that appears in either factorization. So, the smallest number divisible by both 18 and 42 is $2 \times 3^2 \times 7 = \boxed{126}$. The answer is: 126.
LoRA	We want to find the smallest number that is divisible by both 18 and 42. The prime factorization of 18 is 2×3^2 and the prime factorization of 42 is $2 \times 3 \times 7$. To find the smallest number divisible by both 18 and 42, we take the highest power of each prime that appears in either factorization. So, the smallest number divisible by both 18 and 42 is $2 \times 3^2 \times 7 = \boxed{168}$. The answer is: 168.
MiLoRA	We can find the smallest number of marbles by finding the least common multiple (LCM) of 18 and 42. Prime factorizing 18 and 42, we have $18 = 2 \times 3^2$ and $42 = 2 \times 3 \times 7$. The LCM is then $2 \times 3^2 \times 7 = 126$. Therefore, the smallest number of marbles that could be divided up into bags of 18 marbles or 42 marbles is $\boxed{126}$. The answer is: 126.
PiSSA	The smallest number of marbles that could be divided up into bags of 18 marbles is 18, and the smallest number of marbles that could be divided up into bags of 42 marbles is 42. Therefore, the smallest number of marbles that could be divided up into both bags is $18 \times 42 = \boxed{784}$. The answer is: 784.

Table 9: Case Study II for Math Reasoning task on LLaMA2-7B.