HIGH-EFFICIENT DIFFUSION MODEL FINE-TUNING WITH PROGRESSIVE SPARSE LOW-RANK ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

The development of diffusion models has led to significant progress in image and video generation tasks, with pre-trained models like the Stable Diffusion series playing a crucial role. However, a key challenge remains in downstream task applications: how to effectively and efficiently adapt pre-trained diffusion models to new tasks. Inspired by model pruning which lightens large pre-trained models by removing unimportant parameters, we propose **SaRA**, a novel model fine-tuning method with progressive **S**p**a**rse low-**R**ank **A**daptation to make full use of these ineffective parameters and enable the pre-trained model with new task-specified capabilities. In this work, we first investigate the importance of parameters in pre-trained diffusion models and discover that parameters with the smallest absolute values do not contribute to the generation process due to training instabilities. Based on this observation, we propose a fine-tuning method termed SaRA that re-utilizes these temporarily ineffective parameters, equating to optimizing a sparse weight matrix to learn the task-specific knowledge. To mitigate potential overfitting, we propose a nuclear-norm-based low-rank sparse training scheme for efficient fine-tuning. Furthermore, we design a new progressive parameter adjustment strategy to make full use of the finetuned parameters. Finally, we propose a novel unstructural backpropagation strategy, which significantly reduces memory costs during fine-tuning. Our method enhances the generative capabilities of pre-trained models in downstream applications and outperforms existing fine-tuning methods in maintaining model's generalization ability.

1 INTRODUCTION

In recent years, with the development of diffusion models (Ho et al., 2020; Rombach et al., 2022), tasks such as image generation (Van Le et al., 2023; Zhang et al., 2023a), video generation (Guo et al., 2023; Blattmann et al., 2023), and 3D generation (Poole et al., 2022; Sun et al., 2023) have made significant advancements. Pre-trained diffusion models, particularly the Stable Diffusion series (Rombach et al., 2022), have played a crucial role in these developments, including image customization (Van Le et al., 2023), image editing (Kawar et al., 2023), and controllable generation (Zhang et al., 2023a; Mou et al., 2024). Additionally, by leveraging prior information from the image domain, diffusion models have been extended to tasks such as video (Guo et al., 2023; Blattmann et al., 2023) and 3D generation (Poole et al., 2022; Sun et al., 2023). As these applications continue to evolve, a core issue emerges: how to effectively and efficiently fine-tune the foundational pre-trained diffusion models and apply them to new tasks.

Existing fine-tuning methods (Han et al., 2024; Pan et al., 2024; Ansell et al., 2024; Sung et al., 2021) can be categorized into three categories (Fig. 1): ***1)*** **Additive fine-tuning (AFT) methods** (Chen et al., 2022), which introduce additional modules to fine-tune the model, such as adapter-based tuning (Ye et al., 2023; Mou et al., 2024). However, these methods require additional modules and parameters, which has changed the source model, and also introduced additional latency during the inference stage. ***2)*** **Reparameterized fine-tuning (RFT) methods** (Hu et al., 2021; Zhang et al., 2023b), which primarily utilize low-rank matrices to learn new information and merge the learned parameters with the pre-trained one, but it still suffers from the risk of overfitting, since all parameters are adjusted by the low-rank matrices globally. Moreover, the choice of rank and the specific

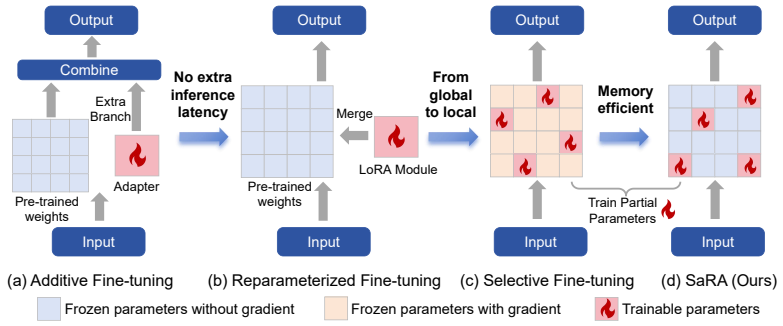

Figure 1: The reparameterized fine-tuning methods (b) address the additional inference latency introduced by additive fine-tuning methods (a) through reparameterizing the pre-trained weights from a global view. Selective fine-tuning methods (c) improve upon global parameter updates by employing sparse updates, which better preserve the model prior by freezing most of the pre-trained parameters. Our SaRA (d) further enhances (c) by significantly reducing memory costs and achieving superior performance in both adaptation capability and prior preservation.

layers to which LoRA is applied requires a tailored design for each model. ***3)*** **Selective-based fine-tuning (SFT) methods** (Guo et al., 2020; Ansell et al., 2021), which select a subset of the model's existing parameters for fine-tuning. However, the complex parameter selection process and high memory cost restrict their application in diffusion models. Overall, both AFT and RFT methods require model-specific designs, *e.g.,* exploration of which layers to apply Adapters or LoRAs within the model, and the hidden dimension or rank needs to be adjusted according to the specific tasks. The SFT method introduces considerable latency, suffers from hyperparameter sensitivity in parameter selection, and also performs poorly in terms of effectiveness and training efficiency. Therefore, a pressing question arises: Can we design a universal method that is ***model-agnostic, does not require hyperparameter searching, inherently avoids overfitting, and simultaneously achieves high-efficiency plug-and-play model fine-tuning***?

Inspired by a theory in model pruning, which posits that within a trained model, there exist parameters with relatively small absolute values that have negligible impact on the model's output, an intuitive idea is: whether we can find a way to leverage these ineffective parameters to make them effective again, and enhance the model's generative capabilities. To achieve this goal, the target "ineffective" parameters we seek must possess two properties: 1) **temporary ineffectiveness**: the parameters themselves have minimal impact on the current model's output; 2) **potential effectiveness**: the parameters are not redundant due to the model structure, but have a certain ability to learn new knowledge (if handled properly, they can be effective again). We first conducted an analysis on the influence of small parameters in pre-trained diffusion models on the model outputs, and found that the smallest 10% (even 20%) of parameters by absolute values did not contribute much to the generative process (Fig. 2, see Sec. 3.1). Furthermore, we examined the potential effectiveness of these parameters and discovered that their ineffectiveness is not inherent (extrinsic) to the model's nature, but rather due to the instability of the training process (see Sec. 3.2). Specifically, the randomness in the training process causes some parameters to approach zero by the end of training. This observation inspired us to rationally utilize these temporally ineffective parameters to make them effective again and fine-tune pre-trained generative models.

Therefore, we propose **SaRA**, a novel fine-tuning method for pre-trained diffusion models that trains **the parameters with relatively small absolute values**. We first identify the "temporally ineffective, potentially effective" parameters as paramters smaller than a threshold in the pre-trained weights. We then efficiently fine-tune these parameters in the pre-trained weights by sparse matrices while preserving prior knowledge. To mitigate the risk of overfitting due to the potential high rank of sparse matrices, we propose a **low-rank sparse training** scheme, which employs a nuclear norm-based low-rank loss to constrain the rank of the learned sparse matrices, achieving efficient fine-tuning of diffusion models. In addition, recognizing that some parameters may not be fully utilized during the fine-tuning process, we propose a **progressive parameter adjustment strategy**, which introduces a second stage to reselect parameters below the pre-defined threshold and train them, ensuring that almost all parameters contribute effectively. Finally, different from the typical selective PEFT methods that retain the gradient of the entire parameter matrices and require high memory cost (the same as full-parameter fine-tuning), we propose an **unstructural backpropagation strategy** with smaller memory cost. In this strategy, we only retain the gradients for the parameters to be

updated, and automatically discard the gradients for other parameters during the backpropagation process. This results in a memory-efficient selective PEFT method, which also advances the development of future selective PEFT techniques. Compared to previous fine-tuning methods (Hu et al., 2021; Valipour et al., 2022; Hayou et al., 2024), our SaRA is capable of effectively enhancing the generative capabilities of the pre-trained model itself, and it also demonstrates the best ability for model adaptation and prior preservation in different downstream tasks.

Contributions of this paper can be summarized in the following four aspects: ***1)***We investigate the importance of the parameters in pre-trained diffusion models, revealing the temporal ineffectiveness and potential effectiveness of the parameters with the smallest absolute weight, which motivates us to make full use of these parameters. ***2)*** We propose SaRA, a novel efficient fine-tuning method based on progressive sparse low-rank adaptation, enabling the model to learn new knowledge without influencing the original generalization ability. ***3)*** We propose unstructural backpropagation, which resolves the high memory consumption problem of selective PEFT methods and surpasses LoRA in memory efficiency (save more than 40% GPU memory than LoRA and selective PEFT methods). ***3)*** We efficiently encapsulated and implemented our method in a single line of code modification, which significantly reduces the coding overhead associated with fine-tuning pre-trained models.

2 Related Works

2.1 Diffusion Models

Diffusion models (Ho et al., 2020; Rombach et al., 2022) have demonstrated significant advantages in image generative tasks. Text-to-image models, represented by Stable Diffusion (Rombach et al., 2022), have diversified into various applications. However, their large parameter sizes somewhat limit the feasibility of full fine-tuning to adapt to specific new tasks. Methods such as ControlNet (Zhang et al., 2023a), T2I-Adapter (Mou et al., 2024), and IP-Adapter (Ye et al., 2023) achieve controlled generation under different conditions by adding external networks to diffusion models. Additionally, models like LoRA (Hu et al., 2021) and DreamBooth (Ruiz et al., 2023) enhance the original diffusion models through fine-tuning, enabling them to generate content in new domains and concepts. Furthermore, some video generation models (Guo et al., 2023; Blattmann et al., 2023) are built on diffusion models to achieve video generations and employ Lora and adapters to accomplish controllable video generations.

2.2 Parameter-efficient model fine-tuning

Addictive Parameter Fine-tuning (AFT). AFT introduces additional modules to the model while keeping the pre-trained backbone fixed. Serial Adapter (Houlsby et al., 2019) enhances the Transformer block by adding new modules after the self-attention layer and FFN layer. AdapterFusion (Pfeiffer et al., 2020) streamlines this by inserting adapter layers only after the FFN layers to boost computational efficiency. Parallel adapters, including Adaptformer (Chen et al., 2022), CoDA (Lei et al., 2023), and KronA (Edalati et al., 2022), reorganize the traditionally sequential adapter layers into a parallel side-network, optimizing both performance and efficiency. To further enhance adapter performance and generalization, multi-task learning strategies like AdaMix (Wang et al., 2022), and Hyperformer (Mahabadi et al., 2021) have also been developed.

Reparameterized Parameter Fine-tuning (RFT). An early work (Aghajanyan et al., 2020) has verified the presence of low intrinsic dimensionality in pre-trained models. LoRA (Hu et al., 2021) proposes to use a low-rank matrix to learn new feature representations. To address the issue of selecting the appropriate rank, DyLoRA (Valipour et al., 2022) employs a dynamic and search-free approach to obtain the optimal rank. AdaLoRA (Zhang et al., 2023b) decomposes the trainable low-rank matrix using singular value decomposition (SVD) and implements dynamic rank adjustment by pruning singular values. Furthermore, numerous subsequent methods (Yang et al., 2023; Ding et al., 2023; Hayou et al., 2024) have aimed to enhance the performance of LoRA.

Selective Parameter Fine-tuning (SFT). Selective parameter finetuning (Han et al., 2024) methods finetune a selected subset of the parameters in the pre-trained model. Diffpruning (Guo et al., 2020) fine-tunes specific parameters by learning a mask matrix, constraining its size through a differentiable L0 norm. PaFi (Liao et al., 2023) selects the parameters with the smallest absolute values for learning, while LTSFT (Ansell et al., 2021), grounded in the Lottery Ticket Hypothesis (Frankle

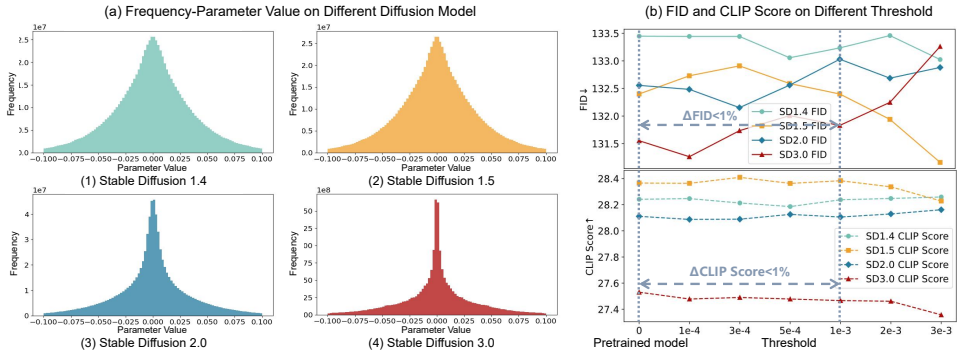

Figure 2: (a) Weight distributions of the pre-trained parameters in Stable Diffusion (SD) 1.4, 1.5, 2.0, and 3.0, which are all similar to a Gaussian distribution, therefore a large number of parameters are around 0. (b) The performance (FID and CLIP Score) of SD Models when the parameters in the pre-trained models with absolute values smaller than a certain threshold are set to 0.

& Carbin, 2018), selects the parameters that change the most during fine-tuning. SHiRA (Bhardwaj et al., 2024) proposes a sparse high-rank adaption method to improve the adaptation ability. Essentially, these methods all learn a sparse mask matrix to fine-tune the pre-trained models.

Overall, both AFT and RFT methods require model-specific designs, e.g., determining which layers to apply Adapter or LoRA, and adjusting the hidden dimension or rank according to specific tasks. Additionally, the SFT method introduces significant latency, exhibits hyperparameter sensitivity in parameter selection, and underperforms in terms of effectiveness and training efficiency. In contrast, our SaRA is model-agnostic, which eliminates the need for layer selection, and can effectively fine-tune the pre-trained model while reducing training costs in both time and memory.

3 THE POTENTIAL EFFECTIVENESS OF THE INEFFECTIVE PARAMETERS

3.1 INEFFECTIVE PARAMETERS IN STABLE DIFFUSION MODELS

Based on the theorem of model pruning proposed in (Liang et al., 2021), which regards the parameters with the smallest absolute values as "ineffective" parameters, we investigated the effectiveness of these parameters in pre-trained stable diffusion models (version 1.4, 1.5, 2.0, and 3.0). We set parameters with absolute values below a certain threshold θ_t (from 10^{-3} to 10^{-5}) to zero, and evaluated the performance of the regularized models on generative tasks with CLIP Score (Radford et al., 2021) and Fréchet Inception Distance (FID) (Heusel et al., 2017).

The results are shown in Fig. 2(b). We observed that within a certain threshold range $\theta_t \in (0, 10^{-3}]$, with the small parameters set to 0, the generative ability of the SD models is minimally affected. And in some cases, the regularized model with "ineffective" parameters set to 0 even outperforms the original model (*i.e.,* no parameters are set to 0, with $\theta_t = 0$). Specifically, SD1.4 and SD1.5 show better FID scores than the original model when thresholds are in the range of $\theta_t \in [5 \times 10^{-4}, 10^{-3}]$, and SD2.0 and SD3.0 exhibit superior FID scores at a threshold of $\theta_t = 10^{-4}$. These results show that parameters with the smallest absolute values have a limited impact on the generative process, and in some cases, they may even slightly impair the model's generative ability.

3.2 UNSTABLE TRAINING PROCESS CONTRIBUTES TO USELESS PARAMETERS

Sec. 3.1 demonstrated that parameters with smaller absolute values have minimal impact on the generative capability of diffusion models. A natural question arises: **are these currently ineffective parameters caused by the model structure and inherently redundant, or are they caused by the training process and can become effective again?** If it is the former case, the structural design of the model prevents these parameters from learning effective information, then these parameters are redundant and unlikely to be useful in subsequent training processes. While if it is the latter case, these parameters are potentially effective when leveraged rationally in the subsequent training. Therefore, we further investigated the reasons behind the ineffectiveness of these parameters, and found that the ineffectiveness is due to the randomness of the optimization process, rather than an inherent inability caused by model structure.

Specifically, we employed a Stable Diffusion model pre-trained on the FFHQ dataset (Karras et al., 2019), whose parameter matrices are denoted as P_0. We recorded the parameters in the pre-trained model with absolute values below the 1% threshold θ_t by a parameter mask M, where $P_M = P_0 \odot M$ denotes the initially below-threshold parameters (1% of all parameters), and $P_{1-M} = P_0 \odot (1 - M)$ denotes the initially above-threshold parameters (99% of all parameters), satisfying:

$$\begin{aligned} |p| &< \theta_t, \forall p \in P_M, \\ |p| &\geq \theta_t, \forall p \in P_{1-M}. \end{aligned} \tag{1}$$

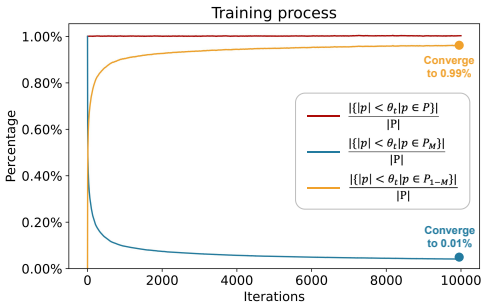

Figure 3: The changes of parameters whose absolute values are bewlow the 1% theshold θ_t during full-parameter fine-tuning. The blue and yellow curves show the proportions of parameters originated from both the initially below-threshold P_M and the initially above-threshold P_{1-M}.

Then, we continue training this pre-trained model on the FFHQ dataset, and observe the changes of its parameters P during the training process. During this fine-tuning stage, we recorded the "source" of parameters whose absolute values are below the threshold $\theta_t(\{|p| < \theta_t\})$, *i.e.,* whether they are initially below-threshold or initially above-threshold. And we found that these parameters originated from both the initially below-threshold P_M and the initially above-threshold P_{1-M}.

The proportions of these two groups and how they change during the finetuning are shown in Fig. 3. As the training progressed, the proportion of P_M remaining below θ_t gradually decreased from 100% to 1% (blue curve decreases from 1.00% to 0.01%); while 1% of the initially above-threshold P_{1-M} eventually fell below θ_t (yellow curve raises from 0.00% to 0.99%). The results indicate that initially ineffective parameters P_M caused by the randomness of the training process, mostly become effective over time (only 1% remaining below threshold). Conversely, some initially effective parameters become ineffective as training continues. This pheonomenon demonstrates that the ineffectiveness of parameters is not inherent to model structure, but rather a result of the stochastic nature of the training process, which causes some parameters to fall below the threshold θ_t at the last training step coincidentally, making them **temporarily ineffective**. As the training continues, most of these parameters regain effectiveness, proving their **potentially effectiveness,** which motivates us to leverage these temporarily ineffective parameters to fine-tune the pre-trained model.

4 Progressive Sparse Low-Rank Model Adaptation

Inspired by the potential effectiveness of parameters with the smallest absolute values, as discussed in Sec. 3, we propose SaRA, a novel parameter-efficient fine-tuning method designed to fully utilize these temporarily ineffective parameters. Specifically, we first identify the ineffective parameters in the pre-trained parameters P_0 by computing a sparse mask $M = P_0 < \theta_t$, where θ_t is a threshold and the sparse mask only selects a small portion from all parameters. We then use this sparse mask to update the initially ineffective parameters $P \odot M$, while keeping the initially effective parameters $P \odot (1 - M)$ frozen. This approach enables the pre-trained model to acquire new capabilities for downstream tasks (through the learnable $P \odot M$) while preserving prior information (through the fixed $P \odot (1 - M)$). To avoid the problem of overfitting caused by strong representation ability due to the potential high rank of the learnable sparse matrix $P \odot M$, we propose a nuclear norm-based low-rank loss to mitigate overfitting (Sec. 4.2). In addition, we propose a progressive parameter adjustment strategy to further make full use of the ineffective parameters by progressively reselecting them (Sec. 4.3). Finally, we propose an unstructured backpropagation strategy, which significantly reduces memory costs and can be applied to enhance all selective PEFT methods.

4.1 Fine-tuning on the potential effective parameters

In Sec. 3, we have demonstrated that parameters with small absolute values are ineffective in the generative process of diffusion models, and this ineffectiveness is not due to the model's architecture but rather the stochastic nature of the optimization process. Therefore, we propose SaRA, which fine-tunes these temporarily ineffective parameters to adapt the pre-trained diffusion model to downstream tasks, enabling it to learn new knowledge while preserving its original generative capability.

Specifically, we first obtain a mask M for the initial parameter set P_0, which satisfies:

$$|p| < \theta_t, \forall p \in P_0 \odot M, \tag{2}$$

where M is a **sparse matrix**, since the theshold θ_t is set low and only selects a small portion from all the parameters. We then use this sparse mask to update the initially ineffective parameters $P_M = P \odot M$, while keeping the initially effective parameters $P \odot (1-M)$ frozen. During training, for the gradient ∇P of the parameters, we use the pre-defined sparse mask M to retain the gradients we need and update the corresponding parameters $P_M = P \odot M$ by:

$$\nabla P_M = \nabla P \odot M + \mathbf{0} \odot (1-M), \quad P_{new} \quad = P - \lambda \cdot \nabla P_M. \tag{3}$$

In this way, we can focus on training the ineffective parameters while keeping the other parameters unchanged, ensuring the original generation ability of the pre-trained model is preserved, while learning new knowledge by the parameters P_M.

4.2 Nuclear Norm-based Low-Rank Constraint

The sparse parameter matrices P_M can sometimes have a high rank, resulting in strong representational capabilities that may lead to overfitting during the training process of downstream tasks. To mitigate this issue, we introduce a nuclear norm-based low-rank constraint on the sparse matrix to prevent the rank from becoming excessively high during the training process.

A direct way to apply low-rank constraint is to minimize the rank of the sparse parameter matrix $Rank(P)$ as a constraint. However, directly minimizing the rank function is computationally intractable due to its non-convex nature. Therefore, we use **nuclear norm** to estimate its rank:

$$\|P_M\|_* = \sum_i \sigma_i(P_M), \text{ where } \sigma_i \text{ are the singular values of } P_M. \tag{4}$$

To compute the nuclear norm $\|P_M\|_*$, we employ the singular value decomposition (SVD) of the matrix $P_M = U\Sigma V^T$, where U and V are orthogonal matrices, and Σ is a diagonal matrix containing the singular values $\sigma_i(P_M)$. The subgradient of the nuclear norm at P_M has been derived by (Watson, 1992), which can be expressed as:

$$\partial\|P_M\|_* = \{UV^T + W \mid W \in R^{m\times n},\, U^T W = 0,\, WV = 0,\, \|W\|_2 \leq 1\}. \tag{5}$$

Based on this derivation of the nuclear norm gradient, we can ensure that gradient descent methods can be employed to incorporate nuclear norm-based low-rank constraints into the training process, thereby achieving our **nuclear norm-based low-rank constrained loss**:

$$L_{rank} = \|P_M\|_* = \sum_i \sigma_i(P_M). \tag{6}$$

4.3 Progressive parameter adjustment

As discussed in Sec. 3.2 and Fig. 3, when continuing training the pre-trained model, the initially ineffective parameters gradually become above threshold and effective, with only 1% of initially ineffective parameters remaining below threshold eventually. However, the speed at which ineffective parameters become effective (the slope of blue curve in Fig. 3) varies during the finetuning process. In the early stage of the finetuning process (*e.g.,* the first 2.5k iterations), a large portion (over 80%) of initially ineffective parameters quickly become effective, with a small part (less than 20%) remaining below threshold. However, the speed slows down in the later stage of finetuning: from 2.5k to 8k iterations, the small portion of remaining below-threshold parameters jumps out of the theshold very slowly. However, the finetuning iterations are typically limited (*e.g.,* a few thousands), in which case the slow speed in the later finetuning stage can cause problems: the remaining below-threshold ineffective parameters may not be trained to be effective and fully utilized.

To address this issue, we propose a **progressive parameter adjustment** strategy. To alleviate the slow speed of ineffective parameters becoming effective in the later stage, this strategy reselects the ineffective parameters that remain below threshold (about 15%-20% of initial ineffective parameters) after the early finetuning stage, and focuses on optimizing these remaining below-threshold parameters in the subsequent finetuning stage. Compared to the finetuning without this reselecting

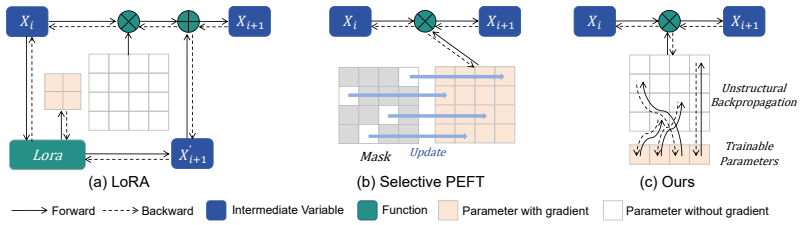

Figure 4: Visualization of our unstructural backpropagation. a) LoRA stores an additional intermediate variable X'_{i+1} in each LoRA layer, and b) selective PEFT methods store the gradients for the whole parameters matrices, causing a waste of memory and computation resources. c) In contrast, our Unstructural Backpropagation method extracts the trainable parameters, sets them as independent leaf nodes, and only retains gradients for them, which largely reduces the memory cost.

operation, this strategy can quickly make remaining ineffective parameters effective again in the later finetuning stage.

Specifically, we introduce a parameter readjustment phase. After the early finetuning stage (we set the first half of the total iterations as the early finetuning stage, *e.g.,* 2,500 iterations when there are 5,000 finetuning iterations) on the initially selected below-threshold parameters P^0_{learn}, we reselect parameters from P^0_{learn} that remain below the predefined threshold as new trainable parameters P_{learn} (which is a subset of P^0_{learn} and typically has 15%-20% of P^0_{learn}'s parameters). Then in the subsequent finetuning stage, we only optimize this subset of initial ineffective parameters, and keep other parameters of P^0_{learn} frozen. By focusing on optimizing the small subset of remaining below-threshold parameters, this strategy greatly improves the speed of ineffective parameters jumping out of the threshold in the later finetuning stage, thereby enhancing the model's adaptation capability. In our experiments, we found that under the same number of finetuning iterations, models without the progressive strategy had 15% of P^0_{learn} remained ineffective after finetuning, while models with the progressive strategy only had 2% of P^0_{learn} that were still ineffective. The results indicate this strategy significantly improves the performance of our method during the fine-tuning process.

4.4 Unstructural Backpropagation

Currently, both the LoRA-based methods (the same for the adapter-based methods) and selective PEFT methods cause a heavy burden on the computation resources: 1) For the LoRA-based methods, since the LoRA module is additional to the original model, there is no need to store the gradients of the model parameters, but they still require additional memory costs to store the intermediate variables in the LoRA module, which is shown in Fig. 4 (a). 2) And for the selective PEFT methods, a persistent issue is that they require the same or even more computational resources (especially GPU memory) as full-parameter fine-tuning. Although they only finetune a subset of the model's parameters for fine-tuning, they retain the gradients of the entire parameter matrices P, because the mainstream deep learning libraries (such as PyTorch and TensorFlow) only support gradient backpropagation and updates for the entire parameter matrices. Consequently, previous selective PEFT methods had to perform gradient backpropagation on the entire parameter matrices P, and then use pre-computed mask matrices M to mask out the gradients of unnecessary parameters by $\nabla P_M = M \odot \nabla P$, and perform an overall parameter update by $P_{new} = P - \lambda \nabla P_M$ (visualized in Fig. 4 (b)). This approach necessitates storing the gradients of all model parameters and the additional mask matrices, leading to greater computational resource demands than full-parameter fine-tuning. This clearly contradicts the "efficient" requirements of PEFT and limits the practical applications of such methods.

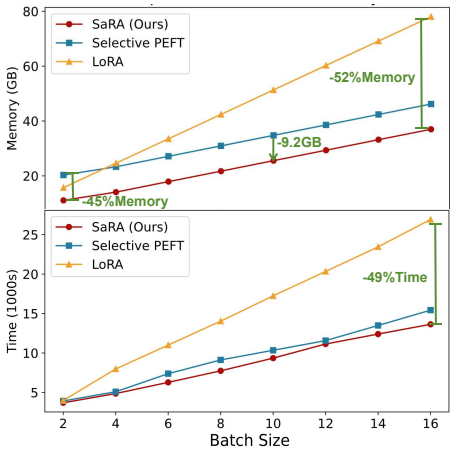

Figure 5: Computation cost on memory and time of different PEFT methods.

To address this issue, we propose **Unstructural Backpropagation** (shown in Fig. 4 (c)), which supports efficient gradient backpropagation and updates for unstructured parameters. Different from

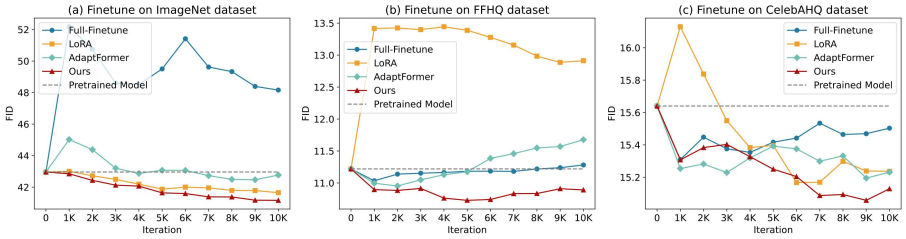

Figure 6: Quantitative comparison among different PEFT methods on Backbone Fine-tuning on ImageNet, FFHQ, and CelebA-HQ datasets. Our method achieves the best FID scores, indicating our method effectively improves the performance of the pre-trained models on the main task.

previous selective PEFT methods that require retaining the gradient for the whole parameter matrices, our Unstructural Backpropagation only needs to retain gradients for the selected subset of below-threshold parameters P_M. Specifically, we first store the mask matrices M corresponding to each layer's parameters that need to be trained[1]. In the computational graph, we deviate from the traditional approach of setting model parameters as leaf nodes. Instead, we extract the trainable parameters $P_{learn} = P[M] \in \mathcal{R}^{\|M\|_0}$ and set them as independent leaf nodes, where $[\cdot]$ denotes element-wise indexing of the matrix. I.e., as shown in Fig. 4 (c), we extract the subset of trainable parameters and combine them into a separate parameter vector, and only retain gradients for this vector. Then, during the forward pass, we define an Unstructural Mapping function $UM(\cdot)$ to update the model parameters P by:

$$P = UM(P, P_{learn}, M), where \begin{cases} P[M] & = P_{learn}, \\ P[1-M] & = P[1-M]. \end{cases} \tag{7}$$

And the updated model paramaters P will then participate in the training process. During backpropagation, we define the Unstructural Backpropagation function $UB(\cdot)$ to propagate the gradients from the model parameters to the trainable parameters by:

$$\nabla P_{learn} = UB(\nabla P, M) = \nabla P[M]. \tag{8}$$

In this way, during backpropagation, the gradients on the model parameters ∇P will be automatically cleared, since it is no longer a leaf node, and only the gradients on the learnable parameters ∇P_{learn} are stored, which significantly reduces the GPU memory during the training process. Notably, unstructual backpropagation is not limited to our method but can be employed in other SFT methods like LT-SFT Ansell et al. (2021), which can advance the development of future SFT fields.

5 Experiments

To validate the effectiveness of our method, we conduct experiments on various tasks, including backbone fine-tuning, downstream dataset fine-tuning, image customization, and controllable video generation (appendix). We compare our method with three state-of-the-art parameter efficient fine-tunining methods: LoRA (Hu et al., 2021), Adaptformer (Chen et al., 2022), and LT-SFT (Ansell et al., 2021); along with the full-parameter fine-tuning method. We evaluate the generation models by three metrics: *1)* Fréchet Inception Distance (FID) (Heusel et al., 2017), *2)* CLIP Score, and *3)* Visual-Linguistic Harmony Index (VLHI), which balances FID and CLIP Score by:

$$VLHI_i = \frac{\max(\{FID_i\}_{i=1}^n) - FID_i}{\max(\{FID_i\}_{i=1}^n) - \min(\{FID_i\}_{i=1}^n)} + \frac{CLIP_i - \min(\{CLIP_i\}_{i=1}^n)}{\max(\{CLIP_i\}_{i=1}^n) - \min(\{CLIP_i\}_{i=1}^n)}$$

5.1 Backbone Fine-tuning

Different from the previous parameter-efficient fine-tuning methods that mainly aim to fine-tune the pre-trained model to downstream tasks, our model enables the pre-trained model to make full use of the parameters. In other words, our finetuning method can improve the performance of pre-trained models on the main task (the original task it is trained on), by optimizing the initially ineffective parameters to be effective and thus increasing the number of effective parameters. Therefore, apart

[1]Since the mask M is of boolean type, it does not consume significant GPU memory.

Backbone	Params	Model	BarbieCore			Cyberpunk			ElementFire			Expedition			Hornify			Mean		
			FID ↓	CLIP ↑	VLHI ↑	FID ↓	CLIP ↑	VLHI ↑	FID ↓	CLIP ↑	VLHI ↑	FID ↓	CLIP ↑	VLHI ↑	FID ↓	CLIP ↑	VLHI ↑	FID ↓	CLIP ↑	VLHI ↑
SD 1.5	50M	LoRA	161.88	**29.93**	1.34	**117.49**	**28.22**	**1.85**	181.66	**27.47**	1.20	136.31	27.39	1.32	156.36	**26.80**	**1.28**	150.74	**27.96**	1.45
		Adaptformer	166.09	29.00	1.00	126.21	27.13	0.66	151.27	26.57	1.29	138.01	26.41	0.63	151.53	26.20	1.18	146.62	27.06	1.18
		LT-SFT	157.80	23.80	0.54	123.59	25.71	0.45	171.67	25.11	0.44	139.29	**27.81**	1.46	158.52	26.35	1.06	150.18	25.76	0.49
		SaRA (Ours)	**148.54**	28.60	**1.75**	121.67	27.30	1.15	**132.67**	26.77	**1.63**	**131.56**	27.34	**1.48**	**140.36**	25.40	1.15	**134.96**	27.08	**1.55**
	20M	LoRA	159.64	**29.65**	1.40	117.21	**28.43**	**1.95**	174.79	**27.61**	1.35	136.38	27.00	1.07	155.85	**27.16**	**1.43**	148.77	**27.97**	1.52
		Adaptformer	159.02	29.08	1.34	123.88	28.07	1.19	174.17	26.53	0.95	137.03	26.67	0.83	157.09	26.63	1.20	150.24	27.39	1.21
		LT-SFT	156.60	23.76	0.59	119.75	25.33	0.70	191.01	25.96	0.49	144.57	**28.01**	**1.37**	165.47	26.89	1.10	155.48	25.99	0.42
		SaRA (Ours)	**153.68**	29.33	**1.63**	**116.69**	28.24	1.94	**138.64**	26.63	**1.50**	**129.98**	27.04	1.36	**145.62**	26.40	1.39	**136.92**	27.53	**1.69**
	5M	LoRA	163.80	**29.93**	1.25	**117.58**	**28.32**	**1.88**	184.99	**27.74**	**1.25**	137.96	27.10	1.07	**153.57**	26.93	**1.40**	151.58	**28.00**	**1.44**
		Adaptformer	164.22	29.37	1.14	120.98	28.11	1.48	184.84	26.66	0.84	143.01	27.35	1.01	171.34	26.85	0.94	156.88	27.67	1.13
		LT-SFT	169.24	24.23	0.08	127.01	25.43	0.03	202.47	26.90	0.68	153.49	**27.96**	0.97	176.41	**27.34**	1.00	165.72	26.37	0.27
		SaRA (Ours)	**153.69**	29.39	**1.64**	118.74	28.17	1.72	**174.86**	**27.04**	1.13	**134.45**	27.06	**1.18**	157.24	26.97	1.33	**147.80**	27.73	**1.44**
	860M	Full-finetune	147.81	27.77	1.65	120.22	27.84	1.47	136.49	25.10	0.95	129.07	26.75	1.21	134.86	24.64	1.00	133.69	26.42	1.30
SD 2.0	50M	LoRA	**157.41**	29.81	1.64	**133.22**	28.00	1.52	187.32	**27.70**	1.29	148.18	27.58	**1.38**	169.92	**26.99**	**1.09**	159.21	**28.02**	1.51
		Adaptformer	161.87	**30.78**	**1.75**	138.02	27.85	1.12	179.44	27.35	1.26	162.45	27.06	0.47	175.39	26.59	0.76	163.43	27.93	1.25
		LT-SFT	164.80	28.13	0.59	134.97	26.40	0.59	183.23	25.90	0.50	153.94	**27.88**	1.33	167.19	26.83	1.08	160.83	27.03	0.57
		SaRA (Ours)	162.72	29.72	1.31	135.05	**28.30**	**1.55**	**151.82**	27.24	**1.68**	**138.77**	26.30	0.96	**165.62**	26.71	1.05	**150.80**	27.65	**1.55**
	20M	LoRA	161.92	30.18	1.52	**129.01**	**28.36**	**2.00**	190.90	27.72	1.24	**147.05**	27.60	**1.44**	168.03	26.97	**1.13**	159.38	**28.16**	1.63
		Adaptformer	**160.29**	**30.42**	**1.70**	141.80	27.92	0.89	190.57	27.33	1.05	157.31	27.07	0.69	175.39	26.59	0.76	165.07	27.86	1.13
		LT-SFT	168.09	28.29	0.47	135.03	26.47	0.62	194.17	26.64	0.66	155.51	**27.88**	1.27	174.64	**27.12**	1.04	165.48	27.28	0.59
		SaRA (Ours)	164.57	30.22	1.39	134.28	28.29	1.60	**163.67**	**27.90**	**1.79**	149.29	27.01	0.98	**165.62**	26.71	1.05	**155.49**	28.03	**1.68**
	5M	LoRA	162.47	29.91	1.39	**132.35**	**28.13**	**1.65**	183.55	**27.68**	1.34	152.69	27.41	**1.09**	**164.00**	26.81	**1.15**	159.01	27.99	1.49
		Adaptformer	**162.25**	30.52	**1.63**	143.41	27.69	0.66	188.42	27.45	1.15	160.23	27.37	0.76	180.07	26.72	0.71	166.88	27.95	1.12
		LT-SFT	175.45	28.74	0.23	137.84	26.55	0.46	209.51	27.29	0.70	161.67	**27.90**	1.03	186.69	**27.62**	1.00	174.23	27.62	0.52
		SaRA (Ours)	165.57	**30.58**	1.47	136.89	27.77	1.15	**174.73**	27.60	**1.45**	**150.89**	27.29	1.09	166.40	26.90	1.13	**158.90**	**28.03**	**1.53**
	866M	Full-finetune	160.87	29.30	1.25	133.19	28.33	1.70	198.45	25.81	0.19	137.84	26.74	1.27	145.99	25.64	1.00	155.27	27.16	0.93
SD 3.0	50M	LoRA	165.22	29.57	1.28	**123.59**	28.38	**1.59**	187.26	27.54	1.36	148.38	26.83	1.50	169.00	26.96	1.35	158.69	27.85	1.52
		Adaptformer	**164.09**	29.81	1.43	126.73	28.23	1.38	186.05	27.14	0.93	156.77	**27.12**	**1.62**	180.11	26.91	1.09	162.75	27.84	1.41
		LT-SFT	209.04	29.45	0.39	158.59	27.72	0.10	208.94	27.03	0.14	204.16	26.02	0.00	189.26	26.06	0.38	194.00	27.26	0.18
		SaRA (Ours)	170.73	**30.63**	**1.73**	128.35	**28.45**	1.50	**179.87**	**27.63**	**1.68**	**137.92**	26.47	1.35	**158.86**	**27.10**	**1.65**	**155.15**	**28.06**	**1.77**
	20M	LoRA	**156.23**	30.18	**1.77**	**123.12**	28.22	**1.48**	187.14	27.76	1.62	**143.59**	**26.96**	**1.68**	174.62	26.63	1.03	**156.94**	27.95	**1.64**
		Adaptformer	174.32	30.30	1.49	128.73	**28.31**	1.39	**175.60**	**27.77**	**1.96**	150.69	26.50	1.19	174.61	26.56	0.99	160.79	27.89	1.50
		LT-SFT	167.02	29.19	1.05	154.04	27.67	0.19	203.23	26.91	0.16	155.68	26.48	1.10	177.42	26.21	0.71	171.48	27.29	0.77
		SaRA (Ours)	166.21	**30.41**	1.70	126.69	28.19	1.35	180.74	27.31	1.28	150.15	26.88	1.52	**163.78**	**27.01**	**1.48**	157.52	**27.96**	**1.64**
	5M	LoRA	161.80	30.14	1.64	**124.17**	28.06	1.33	**174.66**	27.27	**1.40**	**149.85**	27.01	1.63	**172.56**	26.88	1.23	**156.61**	27.87	1.59
		Adaptformer	168.98	30.50	**1.69**	127.35	27.89	1.11	204.69	**27.71**	1.05	158.60	27.03	1.52	182.22	26.88	1.03	168.37	28.00	1.40
		LT-SFT	**158.26**	29.29	1.27	134.81	27.69	0.75	181.68	27.27	1.20	153.52	**27.20**	**1.74**	193.25	26.61	0.63	164.30	27.61	1.20
		SaRA (Ours)	174.42	**30.60**	1.64	125.14	**28.91**	**1.94**	194.79	27.63	1.24	157.20	27.17	1.65	181.39	**27.20**	**1.24**	166.59	**28.30**	**1.68**
	2085M	Full-finetune	162.33	28.69	0.88	151.57	27.59	0.20	174.12	27.16	1.29	135.28	26.09	1.06	144.56	25.58	1.00	153.57	27.02	1.00

Table 1: Comparison with different parameter-efficient fine-tuning methods on Stable Diffusion 1.5, 2.0, and 3.0. For most of the conditions, our model achieves the best FID and VLHI score, indicating that our model learns domain-specific knowledge successfully while keeping the prior information well. **Bold** and underline represent the best and second best results, respectively.

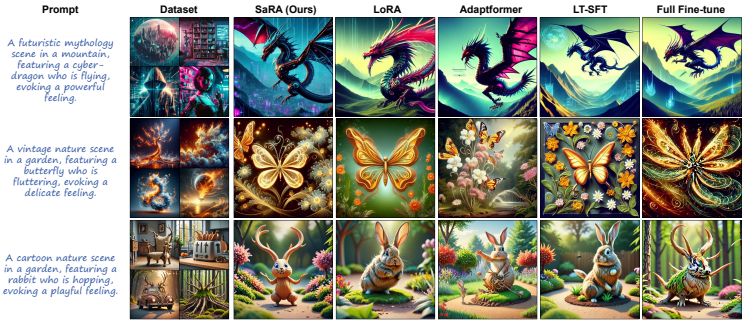

Figure 7: Comparison of the generated images between different PEFT methods.

from experimenting on downstream tasks like traditional PEFT methods, we first apply our method to the main task of the pre-trained model, continuing to fine-tune the backbone on the original training dataset, in order to explore whether our method can enhance the base model's performance. Specifically, we employ the pre-trained Stable Diffusion models on ImageNet (Deng et al., 2009), FFHQ (Karras et al., 2019), and CelebA-HQ (Karras et al., 2017) datasets, and fine-tune them on these pre-trained datasets for 10K iterations. We compare our method with full-parameter finetuning, LoRA, AdaptFormer, and LT-SFT by computing the FID metric between 5K generated data and 5K randomly sampled data from the source dataset. The results are shown in Fig. 6, which demonstrates that our method achieves the best FID scores, indicating our method effectively improves the performance of the pre-trained models on the main task.

5.2 Model Fine-tuning on Downstream Tasks

Downstream Dataset Fine-tuning. In this experiment, we choose 5 widely-used datasets from CIVITAI[2] with 5 different styles to conduct the fine-tuning experiments, which are Barbie Style, Cyberpunk Style, Elementfire Style, Expedition Style and Hornify Style. To comprehensively com-

[2]https://civitai.com/articles/2138/lora-datasets-training-data-list-civitai-dataset-guide

pare PEFT methods, we conduct three sets of experiments for each PEFT method on Stable Diffusion 1.5, 2.0, and 3.0, with selected trainable parameter sizes of 50M, 20M, and 5M. We compute the FID score, CLIP score for the generated data, along with VLHI, which measures both the style (FID) and generalization (CLIP score). The quantitative results are shown in Tab. 1, from which we can draw the following conclusions: ***1)*** Our model can always achieve the best VLHI on average across five datasets, indicating that our model can preserve the prior information in the pre-trained model well (a good CLIP score), while learning as much task-specific knowledge as possible (a good FID), outperforming all the other PEFT methods and full-finetune method; ***2)*** As the number of learnable parameter increases, our model can learn more task-specific knowledge (better FID), but may lose part of the prior information (lower CLIP score); ***3)*** For Stable Diffusions 1.5 and 2.0, our model achieves the best FID and usually the second best CLIP score on average across five datasets, and under different parameter numbers; while for Stable Diffusion 3.0, which has much more parameters than SD 1.5 and 2.0, our model achieves the best CLIP score and usually the second best FID on average across five datasets. The results indicate that for a larger pre-trained model, more learnable parameters are needed to learn the task-specific knowledge well. Moreover, we provide some qualitative comparisons in Fig. 7, which shows the superior generation quality of our method (See appendix for more details.).

5.3 Ablation Studies

We conduct ablation studies to validate the effectiveness of our proposed modules: the progressive parameter adjustment (PPA), and the low-rank constrained loss ($\mathcal{L}_{rank}$). Then, we further assess the effectiveness of training parameters with the smallest absolute values, by comparing different parameter-selection strategies, including selecting the largest parameters and random parameters. We conduct downstream dataset finetuning experiments using the Expedition dataset comparing six ablated models: *1)* model without PPA and $\mathcal{L}_{rank}$, *2)* model with PPA but without $\mathcal{L}_{rank}$, *3)* model with $\mathcal{L}_{rank}$ but without PPA, *4)* model with both PPA and $\mathcal{L}_{rank}$ (Ours), *5)* model fine-tuned with the largest absolute values parameters, and *6)* model fine-tuned with randomly selected parameters. The quantitative metric results are presented in Tab. 2: *1)* The model without both the PPA and $\mathcal{L}_{rank}$ results in a poor FID and low CLIP score. *2)* Introducing PPA improves the FID but decreases the CLIP score, indicating its effectiveness in learning task-specific knowledge. *3)* Incorporating $\mathcal{L}_{rank}$ helps achieve a better CLIP score, but results in a worse FID, indicating its effectivenes in better preserving the model prior knowledge, but with a loss of task-specific information. *4)* Regarding parameter-selection strategies, fine-tuning the largest absolute values parameters yields a relatively good FID but the worst CLIP score, suggesting that fine-tuning the most effective parameters severely disrupts the model's prior knowledge and leads to worse content-text consistency. *5)* Moreover, fine-tuning randomly selected parameters results in both poor FID and CLIP scores, indicating randomly selecting parameters to finetune is unable to learn task-specific knowledge and preserve the model's prior. *6)* In contrast, our model achieves the best VLHI, validating its effectiveness in both fitting capability and prior preservation. More analysis of the hyperparameters is presented in the appendix.

Method	FID ↓	CLIP Score ↑	VLHI ↑
w/o. PPA &$\mathcal{L}_{rank}$	134.75	27.01	1.16
w. PPA, w/o. $\mathcal{L}_{rank}$	<u>130.95</u>	26.66	<u>1.56</u>
w. $\mathcal{L}_{rank}$, w/o. PPA	135.31	<u>27.12</u>	0.89
w. PPA &$\mathcal{L}_{\mathbf{rank}}$ (Ours)	131.56	**27.34**	**1.79**
Tuning Largest Parameters	**130.55**	25.42	1.00
Tuning Random Parameters	133.57	26.58	0.97

Table 2: Ablation studies on six ablated models.

6 Conclusion

In this paper, we propose SaRA, a novel parameter-efficient fine-tuning method, which makes full use of the ineffective parameters with the smallest absolute values in the pre-trained model. We propose a nuclear norm-based low-rank loss to constrain the rank of the learned sparse matrices, thereby avoiding model overfitting. Moreover, we design a progressive parameter adjustment strategy, which can further improve the effectiveness of the fine-tuned parameters. Finally, we propose a novel unstructural backpropagation method, largely saving the memory cost during parameter fine-tuning, which can also reduce the memory costs for other selective PEFT methods. Extensive experiments demonstrate the effectiveness of our method, which achieves the best fitting capability while keeping the prior information of the pre-trained model well.

REFERENCES

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*, 2020.

Alan Ansell, Edoardo Maria Ponti, Anna Korhonen, and Ivan Vulić. Composable sparse fine-tuning for cross-lingual transfer. *arXiv preprint arXiv:2110.07560*, 2021.

Alan Ansell, Ivan Vulić, Hannah Sterz, Anna Korhonen, and Edoardo M Ponti. Scaling sparse fine-tuning to large language models. *arXiv preprint arXiv:2401.16405*, 2024.

Kartikeya Bhardwaj, Nilesh Prasad Pandey, Sweta Priyadarshi, Viswanath Ganapathy, Rafael Esteves, Shreya Kadambi, Shubhankar Borse, Paul Whatmough, Risheek Garrepalli, Mart Van Baalen, et al. Sparse high rank adapters. In *NIPS*, 2024.

Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your latents: High-resolution video synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22563–22575, 2023.

Shoufa Chen, Chongjian Ge, Zhan Tong, Jiangliu Wang, Yibing Song, Jue Wang, and Ping Luo. Adaptformer: Adapting vision transformers for scalable visual recognition. *Advances in Neural Information Processing Systems*, 35:16664–16678, 2022.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. Sparse low-rank adaptation of pre-trained language models. *arXiv preprint arXiv:2311.11696*, 2023.

Ali Edalati, Marzieh Tahaei, Ivan Kobyzev, Vahid Partovi Nia, James J Clark, and Mehdi Rezagholizadeh. Krona: Parameter efficient tuning with kronecker adapter. *arXiv preprint arXiv:2212.10650*, 2022.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. An image is worth one word: Personalizing text-to-image generation using textual inversion. *arXiv preprint arXiv:2208.01618*, 2022.

Demi Guo, Alexander M Rush, and Yoon Kim. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463*, 2020.

Yuwei Guo, Ceyuan Yang, Anyi Rao, Yaohui Wang, Yu Qiao, Dahua Lin, and Bo Dai. Animatediff: Animate your personalized text-to-image diffusion models without specific tuning. *arXiv preprint arXiv:2307.04725*, 2023.

Zeyu Han, Chao Gao, Jinyang Liu, Sai Qian Zhang, et al. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*, 2024.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Lora+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*, 2024.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

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

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Kaiyi Huang, Kaiyue Sun, Enze Xie, Zhenguo Li, and Xihui Liu. T2i-compbench: A comprehensive benchmark for open-world compositional text-to-image generation. *Advances in Neural Information Processing Systems*, 36:78723–78747, 2023.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4401–4410, 2019.

Bahjat Kawar, Shiran Zada, Oran Lang, Omer Tov, Huiwen Chang, Tali Dekel, Inbar Mosseri, and Michal Irani. Imagic: Text-based real image editing with diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6007–6017, 2023.

Tao Lei, Junwen Bai, Siddhartha Brahma, Joshua Ainslie, Kenton Lee, Yanqi Zhou, Nan Du, Vincent Zhao, Yuexin Wu, Bo Li, et al. Conditional adapters: Parameter-efficient transfer learning with fast inference. *Advances in Neural Information Processing Systems*, 36:8152–8172, 2023.

Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International conference on machine learning*, pp. 12888–12900. PMLR, 2022.

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

Baohao Liao, Yan Meng, and Christof Monz. Parameter-efficient fine-tuning without introducing new latency. *arXiv preprint arXiv:2305.16742*, 2023.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*, 2024.

Ilya Loshchilov, Frank Hutter, et al. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 5, 2017.

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

Chong Mou, Xintao Wang, Liangbin Xie, Yanze Wu, Jian Zhang, Zhongang Qi, and Ying Shan. T2i-adapter: Learning adapters to dig out more controllable ability for text-to-image diffusion models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 4296–4304, 2024.

Rui Pan, Xiang Liu, Shizhe Diao, Renjie Pi, Jipeng Zhang, Chi Han, and Tong Zhang. Lisa: Layerwise importance sampling for memory-efficient large language model fine-tuning. *Advances in Neural Information Processing Systems*, 2024.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*, 2020.

Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022.

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

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22500–22510, 2023.

Jingxiang Sun, Bo Zhang, Ruizhi Shao, Lizhen Wang, Wen Liu, Zhenda Xie, and Yebin Liu. Dreamcraft3d: Hierarchical 3d generation with bootstrapped diffusion prior. *arXiv preprint arXiv:2310.16818*, 2023.

Yi-Lin Sung, Varun Nair, and Colin A Raffel. Training neural networks with fixed sparse masks. *Advances in Neural Information Processing Systems*, 34:24193–24205, 2021.

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

Thanh Van Le, Hao Phung, Thuan Hoang Nguyen, Quan Dao, Ngoc N Tran, and Anh Tran. Anti-dreambooth: Protecting users from personalized text-to-image synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2116–2127, 2023.

Yaqing Wang, Sahaj Agarwal, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. Adamix: Mixture-of-adaptations for parameter-efficient model tuning. *arXiv preprint arXiv:2205.12410*, 2022.

G Alistair Watson. Characterization of the subdifferential of some matrix norms. *Linear Algebra Appl*, 170(1):33–45, 1992.

Adam X Yang, Maxime Robeyns, Xi Wang, and Laurence Aitchison. Bayesian low-rank adaptation for large language models. *arXiv preprint arXiv:2308.13111*, 2023.

Hu Ye, Jun Zhang, Sibo Liu, Xiao Han, and Wei Yang. Ip-adapter: Text compatible image prompt adapter for text-to-image diffusion models. *arXiv preprint arXiv:2308.06721*, 2023.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3836–3847, 2023a.

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

A APPENDIX OVERVIEW

We have provided the code for SaRA in the supplementary material. This appendix provides additional analysis and experiments related to SaRA, including:

- More implementation details (Sec. B);
- More comparison results on downstream dataset fine-tuning (Sec. C);
- More comparison results on image customization (Sec. D);
- Comparison experiments on controllable video generation (Sec. E);
- Scaling weight for SaRA parameter (Sec. F);
- Merging different SaRA parameters (Sec. G);
- More ablation studies (Sec. H);
- Analysis on training efficiency (Sec. I);
- Further analysis to understand what SaRA have learned (Sec. J);
- Hyperparameter analysis (Sec. K);
- More analysis on the learned matrix ΔP (Sec. L);
- Limitations (Sec. M).

B MORE IMPLEMENTATION DETAILS

Metrics. We evaluate the generation models by three metrics: *1)* **Fréchet Inception Distance (FID)** (Heusel et al., 2017) to measure the similarity between the generated image distribution and target image distribution, where a lower score indicates better similarity; *2)* **CLIP Score** to measure the matching degree between the given prompts and generated images with a CLIP L/14 backbone (Radford et al., 2021), where a higher score indicates better consistency; *3)* Additionally, since FID and CLIP scores exhibit a certain degree of mutual exclusivity in finetuning a text-to-image model to downstream tasks (*i.e.,* an overfitted model will result in the best FID but the worst CLIP score), we introduce a new metric, the **Visual-Linguistic Harmony Index (VLHI)**, which is calculated by adding the normalized FID and CLIP scores, to balance the evaluation of style (FID) and the preservation of model priors (CLIP score), where a higher score indicates better performance.

Visual-Linguistic Harmony Index (VLHI). We propose VLHI to evaluate both the style and the generalization of each PEFT method, by balancing FID and CLIP Score. For a group of FIDs $\{FID_i\}_{i=1}^n$ and CLIP Scores $\{CLIP_i\}_{i=1}^n$, we compute the normalized FID and CLIP Score as VLHI:

$$VLHI_i = \frac{\max(\{FID_i\}_{i=1}^n) - FID_i}{\max(\{FID_i\}_{i=1}^n) - \min(\{FID_i\}_{i=1}^n)} + \frac{CLIP_i - \min(\{CLIP_i\}_{i=1}^n)}{\max(\{CLIP_i\}_{i=1}^n) - \min(\{CLIP_i\}_{i=1}^n)} \tag{9}$$

For downstream dataset fine-tuning experiments, we regard the methods in one Stable Diffusion version and one dataset as a group.

Dataset details. In the downstream dataset fine-tuning experiments, we choose 5 widely-used datasets from CIVITAI with 5 different styles to conduct the fine-tuning experiments, which are Barbie Style, Cyberpunk Style, Elementfire Style, Expedition Style, and Hornify Style. Each dataset contains about $200 \sim 400$ images, and for each image, we employ BLIP model (Li et al., 2022) to generate its text annotations. The detailed number of images in each dataset is recorded in Tab. 3

Dataset	Barbie	Cyberpunk	ElementFile	Expedition	Hornify
Image Number	316	440	156	396	236

Table 3: The number of images in each dataset.

Training Details. We use AdamW (Loshchilov et al., 2017) optimizer to train the methods for 5000 iterations with batch size 4, with a cosine learning rate scheduler, where the initial learning rate

lr is calculated corresponds to the thresholds θ_t: $lr = 10^{-3} \times e^{-350\theta_t}$ (refer to Sec. K). For the training images and labeled captions, we recaption them by adding a prefix '$name\ style,$' ($name$ corresponds to the dataset name) before each caption, which is a common trick in fine-tuning Stable Diffusion models to a new domain.

Implementation of SaRA. To enable easy implementation of SaRA, we have efficiently encapsulated it, allowing users to perform SaRA-based fine-tuning by modifying just a single line of training code. As shown in Algorithm 1, we integrate SaRA into the optimizer class, so users only need to replace the original PyTorch optimizer with the SaRA optimizer to automatically initiate SaRA training (The code that needs to be modified is highlighted in green.). The learning rate will be automatically assigned based on the threshold θ_t if it is not specified.

Algorithm 1 SaRA Fine-tuning Pseudocode

```
model = Initialize_model()
# optimizer = AdamW(model.parameters())
optimizer = AdamW-SaRA(model, threshold = θ_t)
for epoch = 1 to N do
    for each mini-batch (x, y) do
        y_pred = model(x; θ)
        loss = Loss_Func(y_pred, y)
        loss.backward()
        optimizer.step()
    end for
end for
```

C MORE COMPARISON RESULTS ON DOWNSTREAM DATASET FINE-TUNING

Visualization results. We compare our model with LoRA (Hu et al., 2021), Adaptformer (Chen et al., 2022), LT-SFT (Ansell et al., 2021) and full-parameter finetuning method. We train all methods for 5,000 iterations and use the trained models to generate 500 images based on 500 text descriptions (generated by GPT-4). The quantitative results are presented in the main paper. In this section, we show more qualitative results on Stable Diffusion 1.5, 2.0, and 3.0 with resolutions 512, 768, and 1,024. The results from Stable Diffusion 1.5, 2.0, and 3.0 are shown in Figs. 8- 10 respectively. It can be seen that our model generates images that contain most of the features in the target domain and are well consistent with the given prompts under different datasets. Moreover, to show the generation diversity of Our SaRA, we further generate more images by the trained SaRA weights on Stable Diffusion 1.5, 2.0, and 3.0, where for each SaRA weight, we generate 5 images with the same prompt and different random seeds. The generated results are shown in Fig. 11- 13. It can be seen that SaRA can generate the target-domain images with high diversity, while keeping the semantics consistent with the given prompts, demonstrating a good preservation of the model prior.

More compared methods. In this section, we compare our model with additional state-of-the-art parameter fine-tuning methods on Stable Diffusion 1.5, including DoRA (Liu et al., 2024) and DiffPruning (Guo et al., 2020), which are the representative reparameterized PEFT and selective PEFT approaches, respectively. The comparison results are presented in Tab. 4. The results show that DoRA performs comparably to LoRA, while DiffPruning cannot learn enough tas-specified knowledge, which results in an extremely high FID. In contrast, our model achieves the best performance as evaluated by VLHI, attaining the lowest FID and competitive CLIP score. Moreover, to demonstrate the effectiveness of our SaRA method among various selective PEFT approaches, we compare SaRA with LT-SFT (Ansell et al., 2021), FishMask (Sung et al., 2021), DiffPruning (Guo et al., 2020), and an ablated method that fine-tunes the largest parameters on Stable Diffusion 1.5 with 50M trainable parameters. The qualitative comparison results are shown in Fig. 18. It can be observed that LT-SFT does not learn the target style well in the ElementFire and Horinfy datasets. FishMask tends to generate artifacts as it tunes some effective parameters in the pretrained weights, disrupting part of the model priors. DiffPruning fails to capture task-specific information, resulting in outputs that differ significantly from the target style (despite the fact that we have tried different hyperparameters). Additionally, the ablated model that fine-tunes the largest parameters tends to overfit, similar to the full-parameter fine-tuning model. Since the most important parameters are all fine-tuned, it is prone to overfitting to the target domain, leading to generated images that do not align well with the given prompts. In contrast, our SaRA fits the five datasets well while preserving the model priors, indicating superior performance among the different selective PEFT methods.

More experiments on Stable Diffusion XL. In this section, we present additional comparison experiments on one of the most widely used stable diffusion models, Stable Diffusion XL 1.0 (Podell et al., 2023), capable of generating images at a resolution of 1024×1024. The results, summarized

Backbone	Params	Model	BarbieCore			Cyberpunk			ElementFire			Expedition			Hornify			Mean		
			FID ↓	CLIP ↑	VLHI ↑	FID ↓	CLIP ↑	VLHI ↑	FID ↓	CLIP ↑	VLHI ↑	FID ↓	CLIP ↑	VLHI ↑	FID ↓	CLIP ↑	VLHI ↑	FID ↓	CLIP ↑	VLHI ↑
SD 1.5	50M	DoRA	158.40	29.48	1.43	119.06	28.16	1.49	171.96	**27.67**	1.41	**131.33**	26.94	1.24	150.33	**26.83**	**1.44**	146.22	27.82	1.53
		LoRA	161.88	**29.93**	1.34	**117.49**	**28.22**	**1.62**	181.66	27.47	1.20	136.31	27.39	1.32	156.36	26.80	1.28	150.74	**27.96**	1.45
		Adaptformer	166.09	29.00	1.00	126.21	27.13	0.64	151.27	26.57	1.29	138.01	26.41	0.63	151.53	26.20	1.18	146.62	27.06	1.18
		LT-SFT	157.80	23.80	0.54	123.59	25.71	0.37	171.67	25.11	0.44	139.29	**27.81**	1.46	158.52	26.35	1.06	150.18	25.76	0.49
		SaRA (Ours)	**148.54**	28.60	**1.75**	121.67	27.30	1.02	**132.67**	26.77	**1.63**	131.56	27.34	**1.48**	**140.36**	25.40	1.15	**134.96**	27.08	**1.55**
	20M	DoRA	158.85	29.22	1.37	**116.23**	28.42	**1.78**	169.91	27.33	1.31	133.80	26.86	1.09	148.97	26.82	**1.47**	145.55	27.73	1.51
		LoRA	159.64	**29.65**	1.40	117.21	**28.43**	1.71	174.79	**27.61**	1.35	136.38	27.00	1.07	155.85	**27.16**	1.43	148.77	**27.97**	1.52
		Adaptformer	159.02	29.08	1.34	123.88	28.07	1.11	174.17	26.53	0.95	137.03	26.67	0.83	157.09	26.63	1.20	150.24	27.39	1.21
		LT-SFT	156.60	23.76	0.59	119.75	25.33	0.53	191.01	25.96	0.49	144.57	**28.01**	**1.37**	165.47	26.89	1.10	155.48	25.99	0.42
		SaRA (Ours)	**153.68**	29.33	**1.63**	116.69	28.24	1.69	**138.64**	26.63	**1.50**	**129.98**	27.04	1.36	**145.62**	26.40	1.39	**136.92**	27.53	**1.69**
	5M	DoRA	156.61	29.07	1.45	**113.26**	27.62	**1.74**	178.70	27.57	**1.28**	135.59	26.88	1.02	161.21	**27.34**	1.37	149.07	27.70	1.38
		LoRA	163.80	**29.93**	1.25	117.58	**28.32**	1.65	184.99	**27.74**	1.25	137.96	27.10	1.07	**153.57**	26.93	**1.40**	151.58	**28.00**	**1.44**
		Adaptformer	164.22	29.37	1.14	120.98	28.11	1.33	184.84	26.66	0.84	143.01	27.35	1.01	171.34	26.85	0.94	156.88	27.67	1.13
		LT-SFT	169.24	24.23	0.08	127.01	25.43	0.03	202.47	26.90	0.68	153.49	**27.96**	0.97	176.41	**27.34**	1.00	165.72	26.37	0.27
		SaRA (Ours)	**153.69**	29.39	**1.64**	118.74	28.17	1.52	**174.86**	27.04	1.13	**134.45**	27.06	**1.18**	157.24	26.97	1.33	**147.80**	27.73	**1.44**
	10M	DiffPruning	217.43	31.41	1.00	180.25	28.43	1.00	241.72	27.49	0.91	184.56	28.67	1.00	206.73	28.30	1.00	206.14	28.86	1.00
	860M	Full-finetune	147.81	27.77	1.65	120.22	27.84	1.47	136.49	25.10	0.95	129.07	26.75	1.21	134.86	24.64	1.00	133.69	26.42	1.30

Table 4: Comparison with different parameter-efficient fine-tuning methods (including additional DoRA and DiffPrune) on Stable Diffusion 1.5. For most of the conditions, our model achieves the best FID and VLHI score, indicating that our model learns domain-specific knowledge successfully while keeping the prior information well.

Backbone	Params	Model	BarbieCore			Cyberpunk			ElementFire			Expedition			Hornify			Mean		
			FID ↓	CLIP ↑	VLHI ↑	FID ↓	CLIP ↑	VLHI ↑	FID ↓	CLIP ↑	VLHI ↑	FID ↓	CLIP ↑	VLHI ↑	FID ↓	CLIP ↑	VLHI ↑	FID ↓	CLIP ↑	VLHI ↑
SD XL	50M	DoRA	164.42	**31.76**	**1.77**	126.45	**29.20**	1.76	175.78	28.23	0.74	139.84	**27.60**	1.12	164.53	27.29	0.90	154.20	**28.82**	1.06
		Lora	168.59	31.68	1.51	132.38	28.96	1.26	134.27	27.65	**1.25**	130.37	27.30	1.34	154.78	**27.32**	**1.38**	144.08	28.58	1.45
		Adaptformer	171.33	30.69	1.06	135.74	28.71	0.83	139.71	27.34	0.92	135.68	27.11	0.98	151.20	26.94	1.15	146.73	28.16	1.06
		LT-SFT	165.41	30.20	1.24	131.08	28.65	1.16	140.62	27.48	0.97	126.10	26.97	1.30	150.94	27.11	1.34	142.83	28.08	1.21
		SaRA (Ours)	**162.53**	30.67	1.54	**126.04**	29.01	**1.79**	**129.92**	**28.73**	**2.00**	**124.48**	27.18	**1.51**	**144.28**	26.66	1.18	**137.45**	28.45	**1.71**
	20M	DoRA	165.18	31.41	1.62	**124.22**	28.95	**1.75**	177.07	**28.25**	0.72	138.72	27.64	1.20	163.20	27.28	0.95	153.68	28.71	1.03
		Lora	163.46	31.58	1.77	132.38	28.96	1.26	**139.89**	28.02	1.31	**131.63**	27.52	**1.43**	157.04	27.32	1.27	144.88	28.68	1.46
		Adaptformer	168.54	31.25	1.38	137.61	28.99	0.87	155.14	28.12	0.94	137.73	27.56	1.19	159.13	27.38	1.24	151.63	28.66	1.10
		LT-SFT	178.51	31.44	0.88	131.72	29.01	1.34	149.82	28.01	1.03	140.51	**27.91**	1.30	154.82	27.16	1.21	151.08	28.71	1.16
		SaRA (Ours)	**162.38**	**31.61**	**1.84**	128.55	**29.21**	1.72	142.60	28.22	**1.35**	135.44	27.72	1.39	**153.33**	**27.44**	**1.58**	**144.46**	**28.84**	**1.58**
	5M	DoRA	**166.21**	31.19	**1.49**	**124.68**	29.09	1.81	174.47	28.05	0.66	139.24	27.37	1.00	165.32	27.26	0.83	153.98	28.59	0.94
		Lora	169.38	30.97	1.25	126.76	29.01	1.73	151.41	27.80	0.86	138.03	27.41	1.07	**157.21**	27.01	0.94	148.56	28.44	1.13
		Adaptformer	178.61	30.88	0.71	138.76	29.21	0.92	160.38	27.99	0.72	144.51	27.63	0.94	161.77	26.96	0.68	156.81	28.53	0.76
		LT-SFT	174.77	31.65	1.16	129.10	29.15	1.64	165.69	28.08	0.62	147.41	27.58	0.78	165.84	27.11	0.65	156.56	28.71	0.88
		SaRA (Ours)	174.95	**31.84**	1.20	127.01	**29.33**	**1.92**	**144.27**	**28.40**	**1.41**	**137.07**	**27.66**	**1.28**	158.02	**27.45**	**1.36**	**148.26**	**28.94**	**1.44**
	Full-finetune		2085M	160.72	28.55	1.00	128.94	27.81	0.77	144.56	27.01	0.59	124.59	26.41	1.00	146.60	26.48	0.89	141.08	27.25	0.81

Table 5: Comparison with different parameter-efficient fine-tuning methods on Stable Diffusion XL. For most of the conditions, our model achieves the best FID and VLHI score, indicating that our model learns domain-specific knowledge successfully while keeping the prior information well.

in Tab. 5, demonstrate that our SaRA consistently achieves the best performance on Stable Diffusion XL 1.0, further validating the effectiveness and robustness of our approach.

More evaluation metrics. While the CLIP score (Radford et al., 2021) measures overall similarity between images and text, it may overlook finer details during evaluation. To address this, we incorporate the attribute evaluation metric (denoted as B-VQA) from T2I-CompBench++ (Huang et al., 2023), which assesses the alignment between generated images and input text prompts at a more granular level. The comparison results are presented in Tab. 6, showing that our model achieves the best or second-best B-VQA score in most cases, demonstrating its ability to preserve fine-grained details described in the input text prompts.

D More comparison results on image customization

Image Customization. Image customization aims to learn a common subject from a few images and then apply it to new images. Dreambooth (Van Le et al., 2023) trains the UNet of a diffusion model to bind the target subject to a rare token and then generates images with the specified content based on the rare token. Since Dreambooth requires fine-tuning the UNet network, we compare the performance of full-finetune (original Dreambooth), LoRA, Adaptformer, LT-SDT, and our method in image customization. We compute the CLIP-Text score and CLIP-IMG Score for the generated data, along with VLHI balancing both the two metrics. As shown in Tab. 7, LoRA achieves a high CLIP-IMG score but the lowest CLIP-Text score, indicating a severe overfitting problem. Other PEFT methods, including full-parameter fine-tuning, achieve relatively low CLIP-IMG and CLIP-Text scores. In contrast, our method achieves the best CLIP-Text score, a competitive CLIP-IMG score (only lower than the overfitted LoRA), and the best average VLHI score across three datasets, demonstrating its effectiveness in image customization tasks. We further conduct the qualitative comparison on fine-tuning Dreambooth. As shown in Fig. 14, our method can learn the subject

Backbone	Params	Model	BarbieCore			Cyberpunk			ElementFire			Expedition			Hornify			Mean		
			FID ↓	B-VQA ↑	VLHI ↑	FID ↓	B-VQA ↑	VLHI ↑	FID ↓	B-VQA ↑	VLHI ↑	FID ↓	B-VQA ↑	VLHI ↑	FID ↓	B-VQA ↑	VLHI ↑	FID ↓	B-VQA ↑	VLHI ↑
SD 1.5	50M	DoRA	158.40	0.37	1.16	119.06	0.42	0.58	171.96	0.48	0.96	**131.33**	0.52	1.39	150.33	0.49	1.31	146.22	0.46	1.20
		Lora	161.88	**0.40**	1.26	**117.49**	**0.47**	**1.35**	181.66	0.51	0.89	136.31	0.52	1.23	156.36	0.49	1.15	150.74	0.48	1.22
		Adaptformer	166.09	0.38	0.82	126.21	0.46	0.56	151.27	**0.68**	**1.73**	138.01	0.53	1.22	151.53	0.50	1.37	146.62	**0.51**	1.60
		LT-SFT	157.80	0.38	1.27	123.59	0.46	0.81	171.67	0.46	0.93	139.29	**0.54**	1.35	158.52	**0.51**	1.25	150.18	0.47	1.19
		SaRA (Ours)	**148.54**	**0.40**	**1.84**	121.67	**0.47**	1.05	**132.67**	0.50	1.58	132.54	0.53	**1.48**	**140.36**	**0.51**	**1.73**	**135.15**	0.48	**1.75**
	20M	DoRA	158.85	0.37	1.14	**116.23**	0.44	0.98	169.91	0.48	0.99	133.80	0.52	1.29	148.97	0.49	1.36	145.55	0.46	1.25
		Lora	159.64	**0.40**	**1.36**	117.21	0.47	1.35	174.79	**0.50**	0.97	136.38	0.52	1.22	155.85	0.49	1.23	148.77	0.48	1.29
		Adaptformer	159.02	0.38	1.22	123.88	0.46	0.72	174.17	0.49	0.96	137.03	0.51	0.98	157.09	0.48	1.09	150.24	0.46	1.13
		LT-SFT	156.60	0.38	1.30	119.75	**0.48**	1.40	191.01	0.50	0.73	144.57	**0.54**	1.19	165.47	**0.51**	1.08	155.48	0.48	1.12
		SaRA (Ours)	**153.68**	**0.40**	**1.64**	116.69	0.47	**1.50**	**138.64**	**0.50**	**1.49**	**129.98**	**0.54**	**1.75**	**145.62**	0.50	**1.53**	**136.92**	**0.48**	**1.72**
	5M	DoRA	156.21	0.37	1.21	**113.26**	0.44	1.29	178.70	0.47	0.84	135.59	0.52	1.24	161.21	0.48	1.01	148.99	0.46	1.12
		Lora	163.80	**0.41**	1.25	117.58	0.46	1.27	184.99	0.50	0.83	137.96	0.52	1.17	**153.57**	0.49	**1.22**	151.58	0.48	1.20
		Adaptformer	164.22	0.34	0.64	120.98	0.46	1.03	184.84	0.49	0.82	143.01	0.53	1.11	171.34	0.49	0.79	156.88	0.46	0.94
		LT-SFT	169.24	0.40	0.91	127.01	**0.49**	1.00	202.47	**0.52**	0.61	153.49	**0.56**	1.00	176.41	**0.53**	1.00	165.72	**0.50**	0.94
		SaRA (Ours)	**153.69**	**0.41**	**1.70**	118.74	0.47	**1.34**	**174.86**	0.51	1.00	134.45	0.52	**1.30**	157.24	0.50	1.21	**147.80**	0.48	**1.36**
	Full-finetune	860M	147.81	0.30	1.00	120.22	0.47	1.15	136.49	0.26	0.95	129.07	0.48	1.00	134.86	0.40	1.00	133.69	0.38	1.00

Table 6: Comparison on FID and B-VQA from T2i-compbench++ (Huang et al., 2023) with different parameter-efficient fine-tuning methods on Stable Diffusion 1.5.

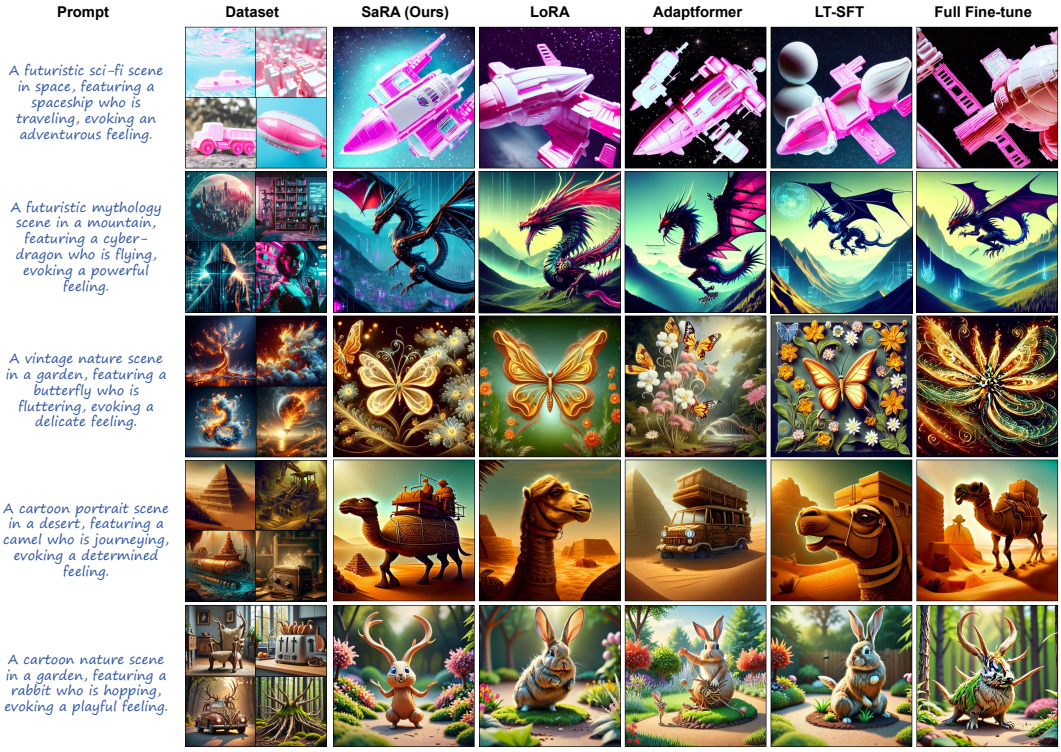

Figure 8: Comparison results between different PEFT methods on Stable Diffusion 1.5.

content well while preserving the prior information of the diffusion model, thereby improving the consistency between the generated images and the given texts, which demonstrates the effectiveness of SaRA in image customization.

E Controllable Video Generation.

We further investigate the effectiveness of our method in fine-tuning video generation models. AnimateDiff (Guo et al., 2023) is a representative video generation model based on Stable Diffusion (Rombach et al., 2022), which inserts temporal attention modules between the original spatial attention modules to model temporal correlations, enabling a diverse text-to-video generation. To achieve more controllable generation, AnimateDiff fine-tunes the temporal attention module using different camera motion data, such as Pan Left, Pan Right, Zoom In, and Zoom Out, to control the camera movements precisely. We compare the effectiveness of various PEFT methods in fine-tuning AnimateDiff for three types of camera movements, including Zoom In, Zoom Out, and Pan Right. Specifically, we collected 1,000 video-text pairs with identical camera movements for each type of camera motion. The temporal attention modules are fine-tuned using full fine-tuning, LoRA, Adaptformer, LT-SDT, and our SaRA. As shown in Fig. 15, the compared methods usually suffer

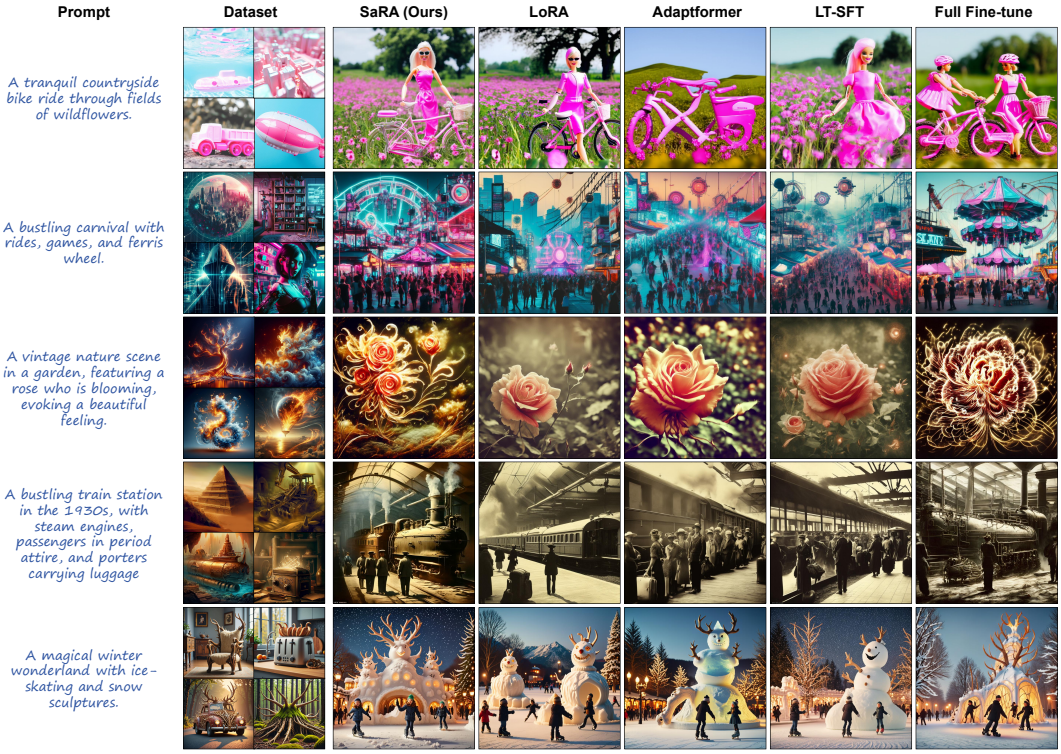

Figure 9: Comparison results between different PEFT methods on Stable Diffusion 2.0.

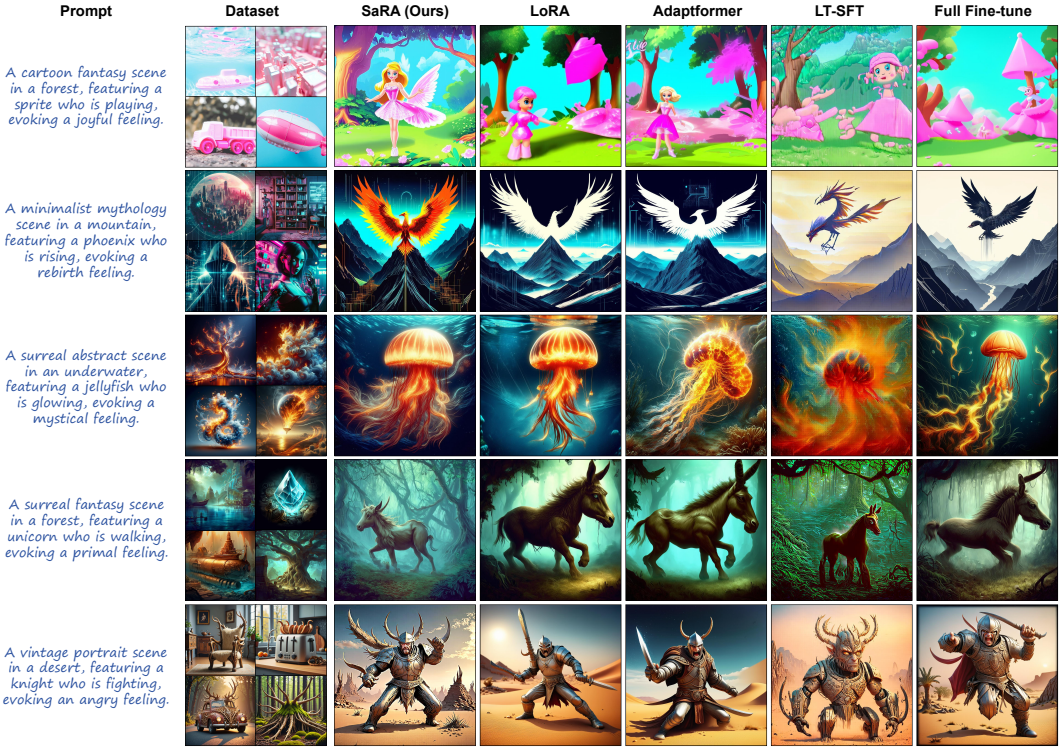

Figure 10: Comparison results between different PEFT methods on Stable Diffusion 3.0.

from generating artifacts in the results (shown in red boxes), indicating that these methods have lost some model priors of specific content during the fine-tuning process. Moreover, for the sea turtle examples, full fine-tuning, LoRA, and LT-SFT exhibit noticeable content degradation. And for the

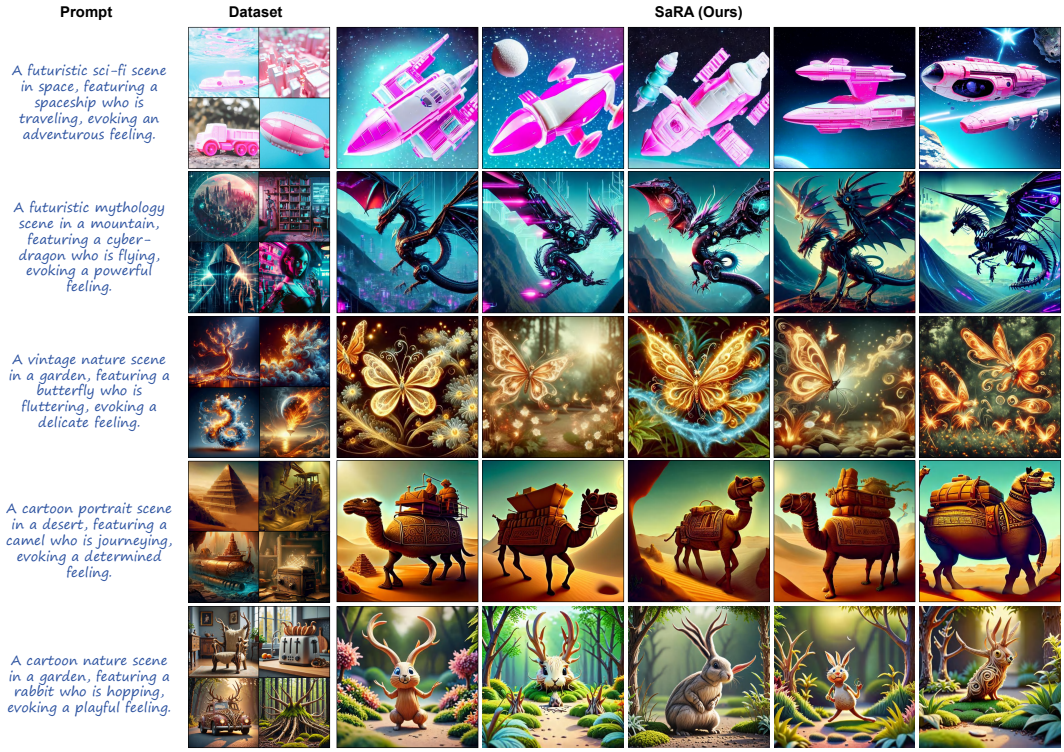

Figure 11: More generation results by SaRA for different downstream datasets on SD 1.5.

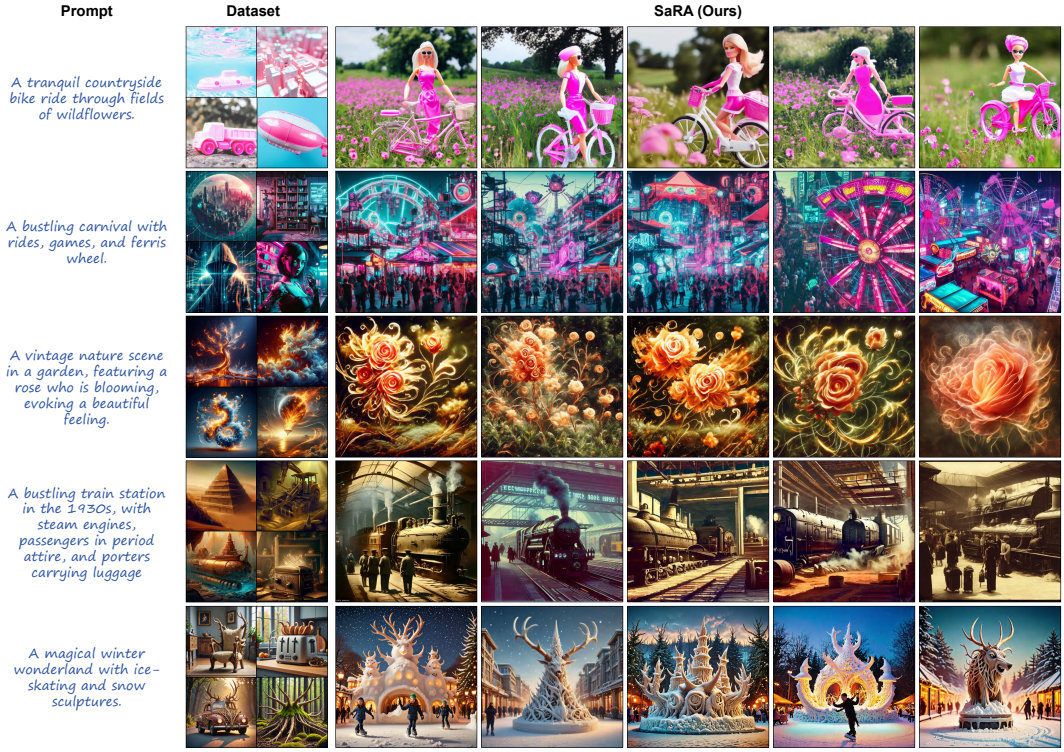

Figure 12: More generation results by SaRA for different downstream datasets on SD 2.0.

pan-right examples, all the compared methods fail to capture the photographer, indicating a significant model overfitting problem. In contrast, our method achieves excellent camera motion control while achieving good consistency between the video content and the text.

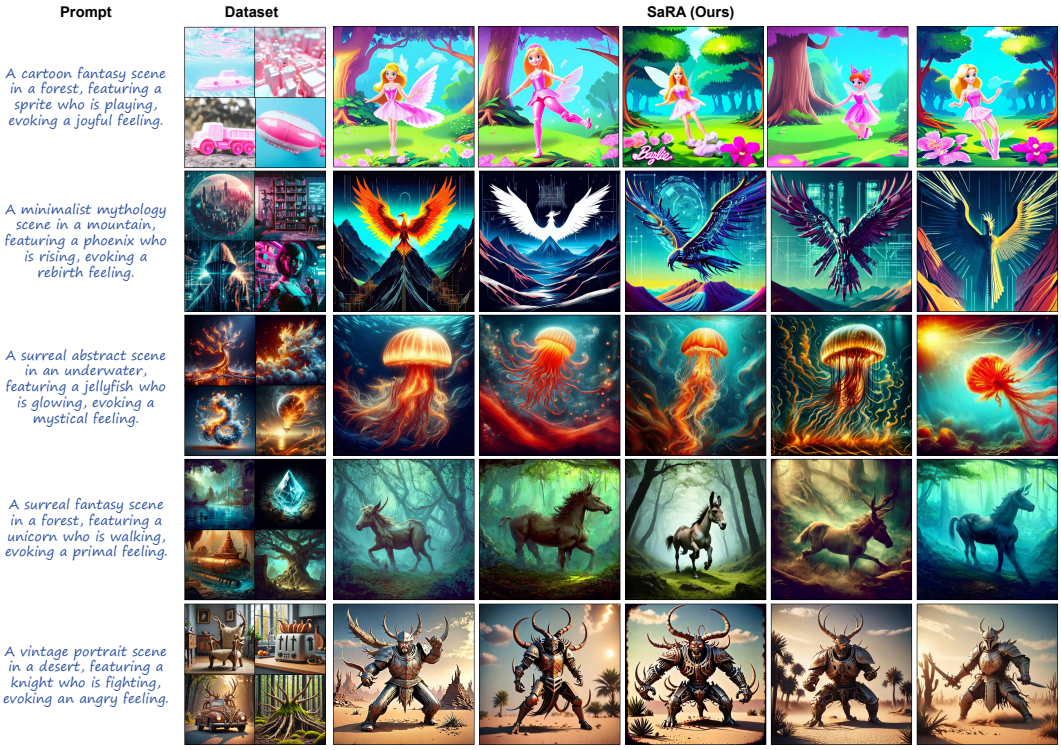

Figure 13: More generation results by SaRA for different downstream datasets on SD 3.0.

Methods	Dog			Clock			Backpack			Mean		
	CLIP-I ↑	CLIP-T↑	VLHI ↑	CLIP-I ↑	CLIP-T ↑	VLHI ↑	CLIP-I ↑	CLIP-T ↑	VLHI ↑	CLIP-I ↑	CLIP-T ↑	VLHI ↑
Textual Inversion	0.788	23.94	0.36	0.789	**24.15**	1.00	0.654	24.09	0.00	0.744	24.06	0.39
Dreambooth + Full Fine-tune	0.776	25.85	1.13	0.894	22.39	1.13	0.856	**25.44**	1.76	0.842	24.56	1.36
Dreambooth + LoRA	**0.895**	23.64	1.00	**0.913**	21.71	1.00	**0.917**	25.23	**1.84**	**0.908**	23.53	1.00
Dreambooth + Adaptformer	0.772	25.42	0.91	0.885	23.18	1.38	0.873	25.25	1.69	0.843	24.62	1.41
Dreambooth + LT-DFT	0.757	23.94	0.13	0.893	22.45	1.14	0.869	25.00	1.49	0.840	23.80	0.79
Dreambooth + SaRA (Ours)	0.790	**25.87**	**1.24**	0.887	23.51	**1.53**	0.886	25.27	1.76	0.854	**24.88**	**1.67**

Table 7: Quantitative comparisons between different PEFT methods on image customization.

F Scaling Weight for SaRA Parameters

Our SaRA aims to learn a sparse low-rank parameter matrix ΔP, which is added to the pre-trained weights P_0. Similar to LoRA (Hu et al., 2021), when applying the learned parameter ΔP to the pre-trained one, we can assign a scaling weight α for the ΔP to control the emphasis extent on the learned target-domain knowledge by:

$$P = P + \alpha \Delta P. \tag{10}$$

We show the results on different α ranging from 0 to 2 on five datasets in Fig. 16. It can be seen that as the scaling weight α increases, the model tends to generate images with more target-domain features, but may lose part of the information specified by the given texts.

G Merging Different SaRA Parameters

For two SaRA parameters ΔP_1 and ΔP_2 learned from two different datasets, we aim to find whether they can be combined to form new parameters that contain the knowledge from both the two datasets. We combine the two parameters by:

$$\Delta P = \alpha_1 \Delta P_1 + \alpha_2 \Delta P_2. \tag{11}$$

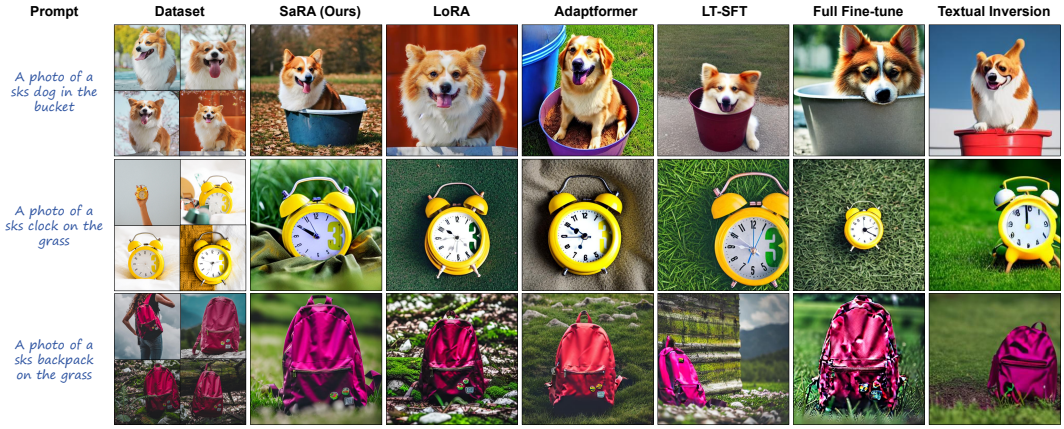

Figure 14: Qualitative comparisons among different PEFT methods on image customization by fine-tuning the UNet model in Dreambooth (Van Le et al., 2023). Our model can accurately capture the target feature while preventing the model from overfitting, outperforming Dreambooth with other PEFT methods and Textual inversion (Gal et al., 2022).

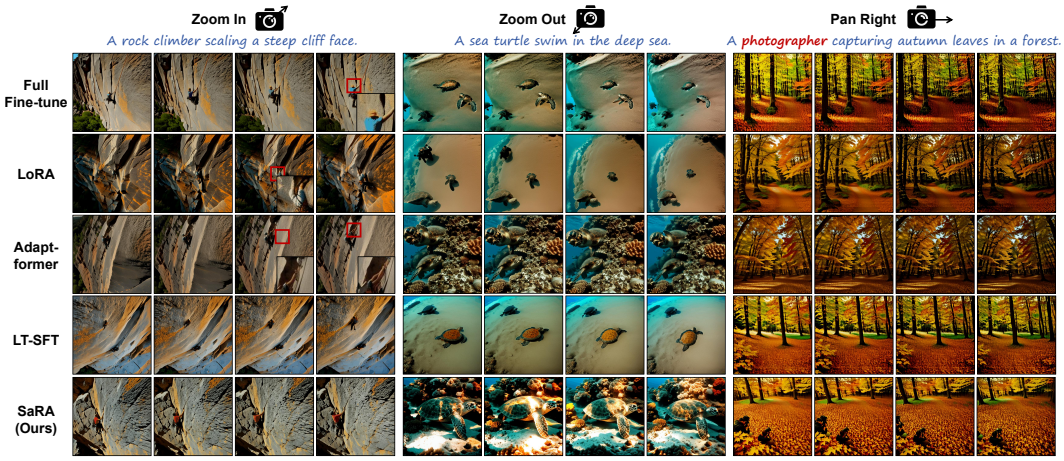

Figure 15: The comparison results of the video generation model (Guo et al., 2023), fine-tuned using different PEFT methods on three video datasets featuring zoom-in, zoom-out, and pan-right camera motions. The red boxes highlight artifacts generated by the compared methods, indicating that these methods have lost some model priors of specific content during the fine-tuning process. In the sea turtle examples, full fine-tuning, LoRA, and LT-SFT exhibit noticeable content degradation. And for the pan-right examples, all the compared methods fail to capture the photographer, indicating significant model overfitting. In contrast, our method achieves excellent camera motion control while preserving video content well.

Then, we employ the combined SaRA parameter ΔP to generate images. We choose four combinations: 'Barbie Style' +'Cyberpunk style', 'Cyberpunk Style' +'ElementFire style', 'ElementFire Style' +'Expedition style', and 'Hornify Style' +'Cyberpunk style', where we simply assign $\alpha_1 = \alpha_2 = 0.6$. The generated images are shown in Fig. 17. It can be seen that, after combining the SaRA parameters learned from two different datasets, the output images contain the features from both two datasets, which indicates that we can merge different SaRA parameters together, enabling more flexible and abundant generation results.

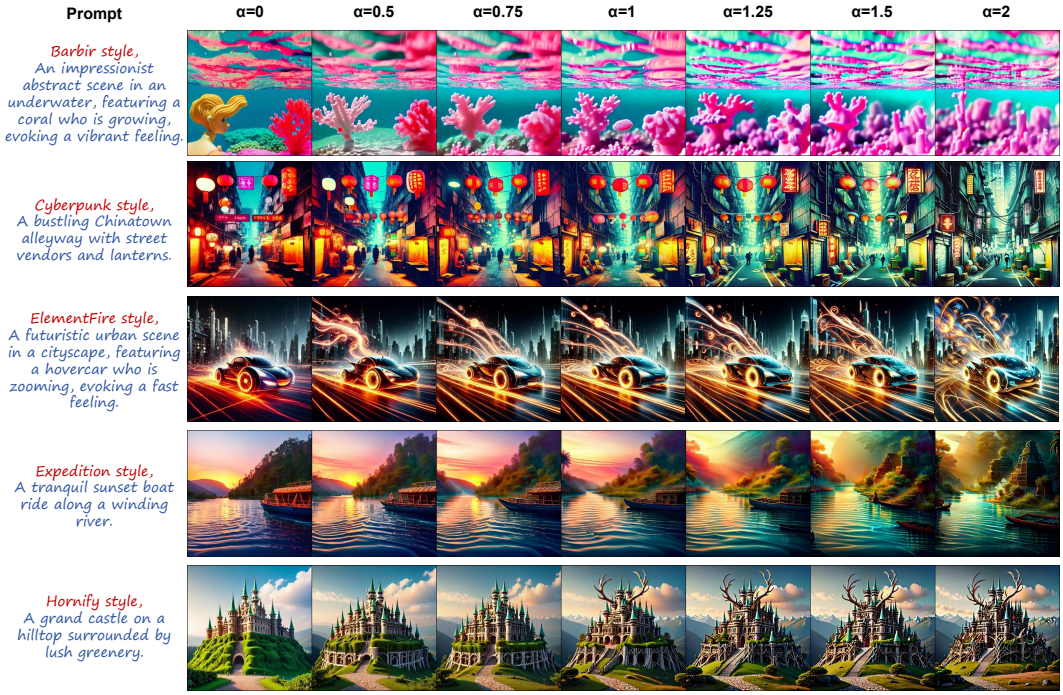

Figure 16: The generated results from different weight α for the learned SaRA parameters by Stable Diffusion 1.5. As α increases, the generated image contains more tar-get domain features.

H MORE ABLATION STUDIES.

Ablation study on threshold. In SaRA, the threshold is an important hyperparameters that influence the size of the parameter space directly. We conduct experiments on different thresholds ranging from 2e-4 to 1e-1 on the Expedition dataset and Stable Diffusion 1.5, and evaluate the generated images by FID and CLIP score. The results are shown in Tab. 8. It can be seen that when the threshold is too small (e.g., 2e-4), the FID becomes much higher, indicating learning less target domain knowledge. And when the threshold is large (i.e., threshold$\leq$2e-3), the model performs quite stably. Since we have a low-rank loss, the model with a high threshold also keeps the CLIP score well. In summary, our SaRA performs well in different thresholds, demonstrating the robustness of our model.

Threshold	2e-4	8e-4	2e-3	5e-3	1e-2	5e-2	1e-1
FID ↓	134.45	129.98	132.54	131.05	130.42	130.71	129.88
CLIP ↑	27.06	27.04	27.38	27.21	27.15	27.04	27.02

Table 8: Ablation study on the threshold.

More ablation study on the low-rank loss. In this section, we conduct more ablation studies on the low-rank loss $\mathcal{L}_{rank}$, where we choose Stable Diffusion 1.5 and two additional datasets (Cyberpunk and ElementFire) for the experiment. The results are shown in Tab. 9, where we can see that the mode without $\mathcal{L}_{rank}$ always tends to get a worse CLIP score, indicating a significant performance drop. Therefore, the low-rank loss is quite necessary in our model to keep the model prior.

I ANALYSIS ON TRAINING EFFICIENCY

In Sec.4.4 of the main paper, we propose unstructural backpropagation, which allows selective PEFT to store and update only the gradients of trainable parameters, significantly reducing memory usage

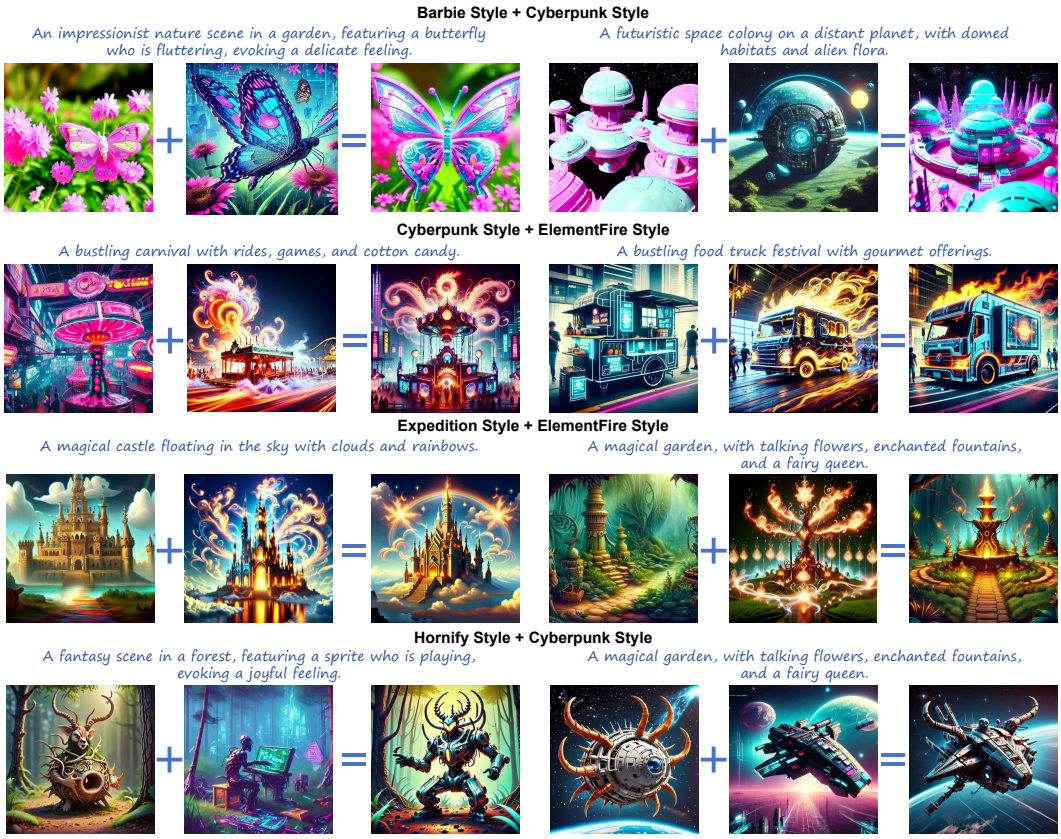

Figure 17: Combining the SaRA parameters learned from two different datasets, the model can generate images with features from both datasets. We show the combination results for 'Barbie Style' +'Cyberpunk style', 'Cyberpunk Style' +'ElementFire style', 'ElementFire Style' +'Expedition style', and 'Hornify Style' +'Cyberpunk style' in this figure.

method	Cyberpunk		ElementFire	
	FID	CLIP	FID	CLIP
SaRA	121.67	27.30	132.67	26.77
SaRA w/o. $\mathcal{L}_{rank}$	120.33	26.52	131.56	25.88

Table 9: More ablation studies on the low-rank loss $\mathcal{L}_{rank}$.

during training. We conducted experiments on the Stable Diffusion 2.0 model using an 80G NVIDIA A100 GPU, comparing the memory usage and training time of LT-SFT (Selective PEFT method), LoRA, and our method across different batch sizes. The results, shown in Fig. 5 of the main paper, demonstrate that our method achieves the lowest memory consumption and training time under all batch sizes. Compared to LT-SFT, we reduce memory usage by a fixed 9.2G (equivalent to the total gradient size of fixed parameters) and achieve over 45% memory reduction for smaller batch sizes. Furthermore, compared to LoRA, our method saves over 52% memory and 49% training time for larger batch sizes, showcasing the efficiency of our SaRA in model fine-tuning.

J Further Analysis to Understand what SaRA have learned

The Correlation between ΔP and P. We further investigate what exactly is learned by the sparse parameter matrix ΔP_{Ours} obtained through our method. Firstly, we examine the relationship between ΔP and the pre-trained parameter matrix P. We want to know whether ΔP has learned new

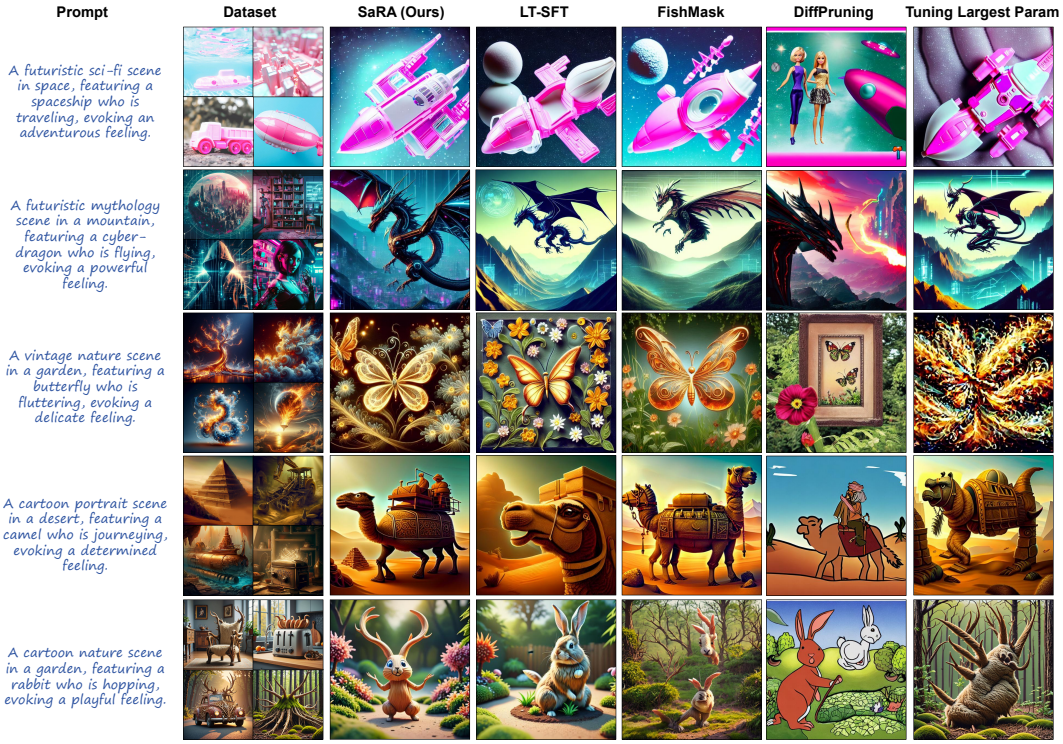

Figure 18: More qualitative comparison with the existing Selective PEFT methods (LT-SFT (Ansell et al., 2021), FishMask (Sung et al., 2021), DiffPruning (Guo et al., 2020)), and the ablated model that fine-tunes the largest parameters) on SD 1.5.

knowledge that is not present in P, or it amplifies some existing but previously not emphasized knowledge in P. To answer this question, we study the subspaces of ΔP and P. We first conduct SVD decomposition on ΔP_{Ours}, and obtain the left and right singular-vector matrices $U_{\Delta P_{Ours}}$ and $V^T_{\Delta P_{Ours}}$. We then project P into the first r-dimensional subspace of ΔP using $U_{\Delta P_{Ours}} P V^T_{\Delta P_{Ours}}$. We quantify the correlation between P and the first r-dimensional subspace of ΔP by calculating the Frobenius norm of this projection $\|U_{\Delta P_{Ours}} P V^T_{\Delta P_{Ours}}\|_F$, where a smaller norm indicates lower correlation between the subspace of ΔP and P.

For a valid reference, we further decompose the parameter matrix ΔP_{LoRA} learned by LoRA using SVD to obtain the respective $U_{\Delta P_{LoRA}}$ and ${V_{\Delta P_{LoRA}}}^T$ matrices, and project the pre-trained parameter matrix P into the first r-dimensional subspace of ΔP_{LoRA} using $U_{\Delta P_{LoRA}} P {V_{\Delta P_{LoRA}}}^T$.

In addition, we calculate an amplification factor to determine how much the parameter matrix ΔP amplifies the directions that are not emphasized by P. The amplification factor is computed as $f_a = \frac{\|\Delta P\|_F}{\|UPV^T\|_F}$. The higher the amplification factor is, the more task-specific knowledge is learned.

We investigate the relationship between the first $r = 4, 16, 64$ dimensional subspaces of ΔP and P. The results are shown in Tab. 10[3], from which we can draw the following conclusions:

1. The learned sparse matrix ΔP_{Ours} from our model has a significant amplification factor, such as ***25.72*** *times for $r = 4$, which indicates the correlation between the first 4-dimensional subspace of ΔP_{Ours} and P is low, and ΔP_{Ours} primarily amplifies the directions that are not emphasized in P.*

2. Compared to the low-rank parameter matrix ΔP_{LoRA} learned by LoRA, our model achieves a higher amplification factor across different values of r, indicating that our method can learn more knowledge that is not emphasized in P than LoRA.

3. As r increases, the amplification factor gradually decreases, suggesting that the knowledge learned by ΔP is mostly contained within P, and the primary role of ΔP is to amplify some of

[3] $\|\Delta P_{ours}\|_F = 4.40$ and $\|\Delta P_{LoRA}\|_F = 4.62$.

Rank	r=4			r=16			r=64		
Matrices	ΔP_{Ours}	ΔP_{LoRA}	P	ΔP_{Ours}	ΔP_{LoRA}	P	ΔP_{Ours}	ΔP_{LoRA}	P
$\|UPV^T\|_F$	0.17	0.34	9.36	0.48	1.14	14.82	2.68	3.90	23.91
Amplification	**25.72**	13.45	-	**6.50**	4.05	-	**1.64**	1.18	-

Table 10: The correlation between the learned parameter matrices ΔP and the pretrained weights P. Our learned parameter matrix ΔP_{Ours} amplifies the directions that are not emphasized in the pretrained weights P, and has a larger amplification factor than LoRA, indicating our model learns more task-specific knowledge than LoRA.

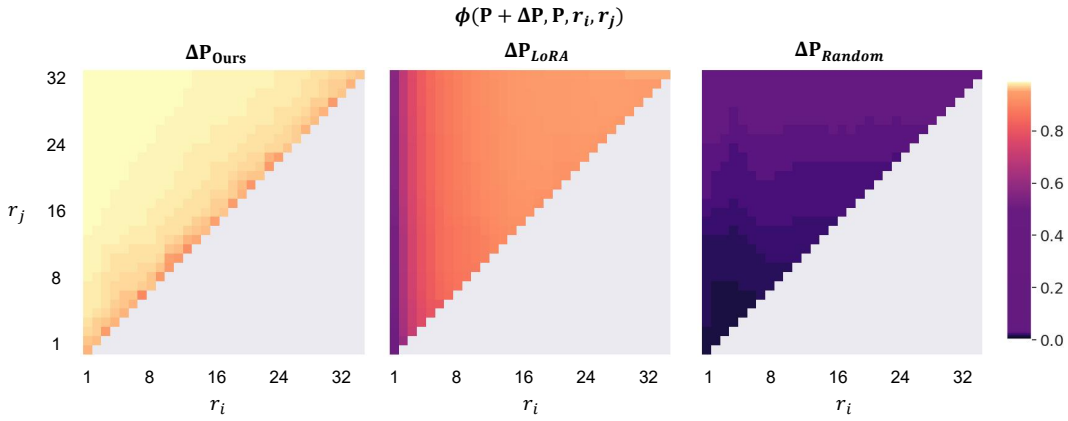

Figure 19: Subspace similarity between $P + \Delta P$ and P. The similarity $\phi(P + \Delta P_{Ours}, P, r_i, r_j)$ between our updated parameter matrix and the pretrained parameter matrix achieves a similarity larger than **96%** across different dimensions of the subspace (r_i, r_j). The results indicare the learned sparse parameter matrix ΔP of our model keeps the prior information in the pretrained parameters P well.

the existing but previously not emphasized knowledge in P (if new knowledge that is not present in P has been learned by ΔP, the correlation should remain low as r increases, and the amplification factor should remain high, which is not the case presented in our experiments).

The Correlation between $P + \Delta P$ and P. We aim to further understand whether our learned parameter matrix ΔP_{Ours} disrupts the information in the original parameter space (spanned by the pretrained weights P), which may lead to overfitting and loss of prior information. To analyze the preservation of prior information, we calculate the correlation between the final updated parameter matrix $(P + \Delta P)$ and the pretrained weights P. Specifically, we calculate the similarity between the subspaces of $(P + \Delta P)$ and P. We decompose $(P + \Delta P)$ and P using Singular Value Decomposition (SVD) to obtain the left-singular unitary matrices U, and examine the similarity between the subspaces spanned by the first r_i singular vectors of $U_{P+\Delta P}$ and the first r_j singular vectors of U_P. We quantify the subspace similarity using the normalized subspace similarity based on the Grassmann distance (Hu et al., 2021):

$$\phi(P_1, P_2, r_i, r_j) = \frac{\|U_1^{r_i T} U_2^{r_j}\|_F^2}{min(r_i, r_j)} \in [0, 1], \quad \text{where } U_k \Sigma_k V_K^T = SVD(P_k), k = \{1, 2\}. \tag{12}$$

We calculate the similarity between the pre-trained parameter matrix P and the updated parameter matrices obtained from three approaches: 1) our model $(P + \Delta P_{Ours})$, 2) LoRA $(P + \Delta P_{LoRA})$, and 3) a random parameter matrix added to the pre-trained parameters $(P + \Delta P_{Random})$. The results are shown in Fig. 19. As a reference, the subspace similarity $\phi(P+\Delta P_{Random}, P, r_i, r_j)$ of randomly updated parameters approaches zero across different dimensions r_j and r_j, indicating random weights will destroy the prior information in the pre-trained weights absolutely. In contrast, the similarity $\phi(P+\Delta P_{Ours}, P, r_i, r_j)$ between our learned parameter matrix added to the pre-trained parameters and the original parameter matrix exceeds 96% across different subspace dimensions

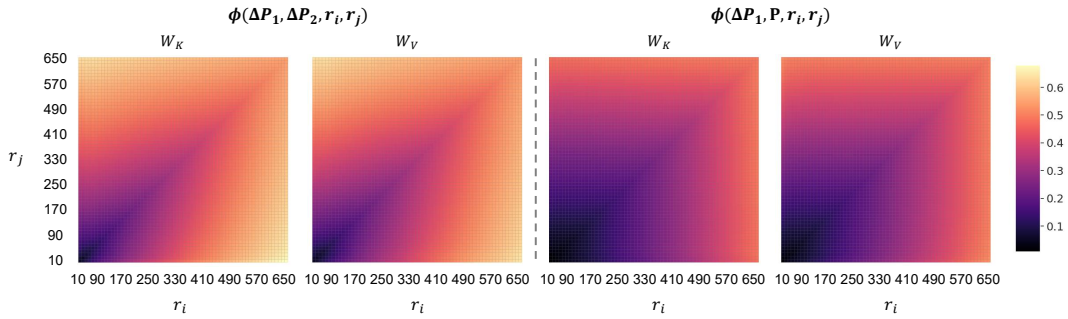

Figure 20: Subspace similarity between ΔP_1 and ΔP_2 under thresholds $2e-3$ and $8e-4$. The total similarity between their learned matrices exceeds 60% (when r is around 650), compared to only 40% similarity between ΔP_1 and the pre-trained weights P, demonstrating the learned matrices from different thresholds learn similar task-specific knowledge, while emphasizing different directions (relative smaller similarity when r is small).

(r_i, r_j), indicating that our learned parameter matrix effectively preserves the information in the original parameter matrix. In addition, compared to the parameter matrix $(P+\Delta P_{Lora})$ updated by LoRA, our updated parameter matrix $(P + \Delta P_{Ours})$ shows greater subspace similarity with the pre-trained parameters P, demonstrating that our model's learned sparse parameter matrix better preserves the prior information of the pre-trained parameters, effectively avoiding model overfitting. Combining this with the conclusions from the previous section, we can further conclude that ***Our model can learn more task-specific knowledge, while more effectively preserving the prior information of the pre-trained parameter matrix than LoRA.***

The Correlation between ΔP under Different Thresholds. We further investigate the relationship between the learned parameter matrices ΔP under different thresholds. Our experiments focus on the matrices for Key W_K and Value W_V from the medium block's attention modules in SD1.5. In this experiments, we select two thresholds, $2e-3$ and $8e-4$ (corresponding to ΔP_1 and ΔP_2), and compute the similarity of their subspaces using Eq. (12). For comparison, we also calculate the similarity between the parameter matrix ΔP_1 learned with a threshold of $2e-3$ and the pre-trained parameter matrix P. The results are shown in Fig. 20. It can be observed that the overall similarity between the parameter matrices learned under the two thresholds exceeds 60% (peaking at around $r = 650$), indicating that the knowledge learned under different thresholds is roughly similar, but with different emphases (lower similarity at smaller r). In contrast, the similarity between ΔP_1 and the pre-trained parameters P is consistently below 40%. The higher similarity between ΔP_1 and ΔP_2 suggests that the learned parameter matrices from different thresholds indeed capture similar task-specific knowledge, which supports the feasibility of fine-tuning the model with fewer parameters.

K Hyperparameter Analysis

In this section, we conduct experiments on different hyperparameters in our model: learning rate, progressive iteration (the iteration for progressive parameter adjustment), and the weight for rank loss λ_{rank}. We chose Stable Diffusion 1.5 and Expedition dataset for the following experiments (if not specified, the threshold $\theta_t = 2e-3$) and evaluated the results by FID, CLIP Score, and VLHI.

Learning Rates and Thresholds. We first investigate the two most critical hyperparameters: learning rate and threshold. We selected learning rates $\{1e{-}4, 2e{-}4, 5e{-}4, 8e{-}4, 1e{-}3\}$ and thresholds $\{2e-4, 5e-4, 8e-4, 2e-3, 5e-3\}$ for our experiments, resulting in a total of 25 models. The quantitative results are shown in Fig. 21. It can be observed that, for the same learning rate, as the threshold increases, the model's FID gradually decreases while the CLIP Score gradually increases. This indicates that a larger learnable parameter set can learn more task-specific information, but is also more likely to lose pre-trained prior knowledge. For the same threshold, increasing the learning rate yields similar results. However, for relatively large thresholds (e.g., $5e-3$), a high learning rate (e.g., $8e-4$ and $1e-3$ in the figure) may cause the model training to collapse. Therefore, selecting an appropriate learning rate is crucial for achieving good results.

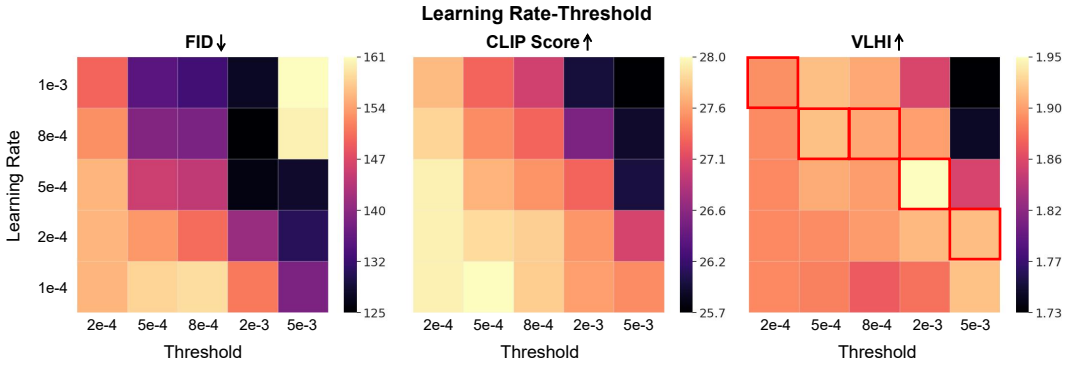

Figure 21: The comparison results on different learning rates and thresholds. The model with a larger threshold should employ a larger learning rate to learn the target-domain information well (red boxes indicate the best results).

We further use the VLHI metric to analyze the performance of the models trained with different learning rates under various thresholds by balancing FID and CLIP scores, as shown in the third column in Fig. 21. The optimal learning rate for each threshold is marked with a red box. It can be seen that as the threshold increases, a gradually decreasing learning rate should be used to prevent severe overfitting. Conversely, as the threshold decreases, a larger learning rate should be employed to enhance the model's ability to learn task-specific knowledge. In summary, there is a negative correlation between the learning rate and the threshold. To adaptively select an optimal learning rate, we fit an exponential function $f(x) = a \times e^b$ using the five data points shown in the figure. The resulting function for adaptively computing the learning rate for different thresholds is shown in Fig. 22. The curve fits the five data points well, and when the threshold approaches 0, the learning rate is approximately $1e-3$, which does not result in an excessively high learning rate. Similarly, for larger thresholds (e.g., $1e-2$), the learning rate is around $3e-5$, comparable to the learning rate used in full fine-tuning, avoiding an excessively low learning rate. We do not consider even larger thresholds, as these parameters are highly effective in the model, and fine-tuning them would contradict the purpose of our method. Therefore, we derive a function to adaptively compute a good learning rate Lr based on the threshold θ_t:

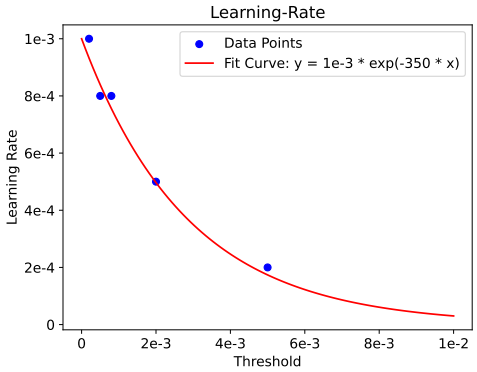

Figure 22: The fit curve of the best pairs of learning rate and threshold.

$$Lr = 10^{-3} \times e^{-350\theta_t}. \tag{13}$$

Learning Rates and Progressive Iteration. We then study the effects of learning rate and progressive iteration (the iteration for progressive parameter adjustment) together. We train the models with learning rates $\{1e-4, 2e-4, 5e-4, 8e-4, 1e-3\}$ and progressive iteration $\{1000, 2000, 2500, 3000, 4000\}$, which forms 25 models in total. The quantitative results are shown in Fig. 23. For all the metrics (FID, CLIP Score, and VLHI), the brighter the color is, the better the model performs. It can be seen that, as the learning rate or progressive iteration grows, the model learns more task-specific knowledge (a better FID), while the CLIP score becomes worse. Therefore, we should balance both the learning rate and progressive iterations, where the model with learning rate $5e-4$ and progressive iteration 2000 achieves the best VLHI, reaching both a good FID and CLIP Score.

λ_{rank} and Progressive Iteration. We then analyze the influence of the weight for rank loss λ_{rank} and progressive iteration at the same time. We train the models with λ_{rank} $\{1e-4, 5e-4, 1e-3, 5e-3, 1e-2\}$ and progressive iteration $\{1000, 2000, 2500, 3000, 4000\}$, which constitutes 25 models in total. The quantitative results are shown in Fig. 24. It can be seen that as λ_{rank} increases,

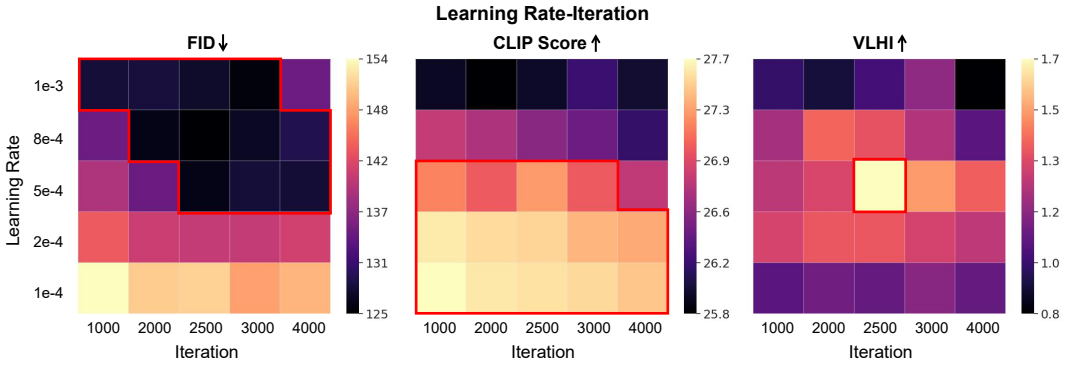

Figure 23: The comparison results on different learning rates and progressive iterations. A larger learning rate or progressive iteration improves the FID while sacrificing the CLIP Score.

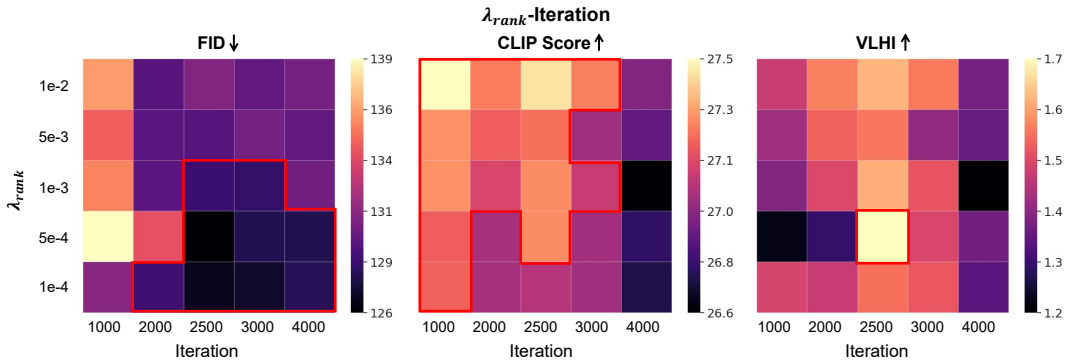

Figure 24: The comparison results on different weights λ_{rank} for rank loss and progressive iterations. A larger λ_{rank} improves the CLIP Score while sacrificing the FID.

the FID becomes worse while the CLIP Score performs better, demonstrating that λ_{rank} helps the model keep the prior information in the pre-trained weights, but with a less effect in fitting the target domain. Therefore, to simultaneously reach a relatively good FID and CLIP Score, we choose $\lambda_{rank} = 5e-3$ with progressive iteration 2500, which results in the best VLHI.

L More Analysis on the Learned Weight Matrix ΔP

The Correlation between ΔP and P under Different Thresholds. We compute the subspace similarity between the learned matrices $Delta_P$ under different thresholds and the pre-trained weights P by Eq. (6) of the main paper. The results are shown in Fig. 25. It demonstrates that ΔP does not contain the top singular directions of W, since the overall similarity between the singular directions in the learned matrices ΔP and the top 32 directions of P is barely around 4%. And it further validates that the matrices ΔP contain more task-specific information rather than repeating the directions that are already emphasized in the pre-trained weights. Moreover, by comparing the ΔP from different thresholds, we can find that as the thresholds grow, the subspace similarity between ΔP and P becomes smaller, indicating that a larger threshold can learn more task-specific information, therefore a large threshold can contribute to a better FID as shown in Tab. 1 of the main paper.

More Analysis on ΔP from Different Layers. In the main paper, we have analyzed the subspace similarity between the learned matrices ΔP from different thresholds, and concluded that the matrices from different thresholds learn similar task-specific knowledge, but emphasize different directions. To further validate this conclusion, we conduct more quantitative analysis between different thresholds ($\theta_t = 2e-3$ and $8e-4$) from the attention layers in the bottom, medium, and up blocks. Moreover, we take all the learnable matrices in the attention module into consideration, including the Query, Key, Value, and FFN matrices (corresponds to W_Q, W_K, W_V, and W_{Out} respectively). The results can be referred to in Fig. 26 the x-axis and y-axis represent $\theta_t = 2e-3$ and

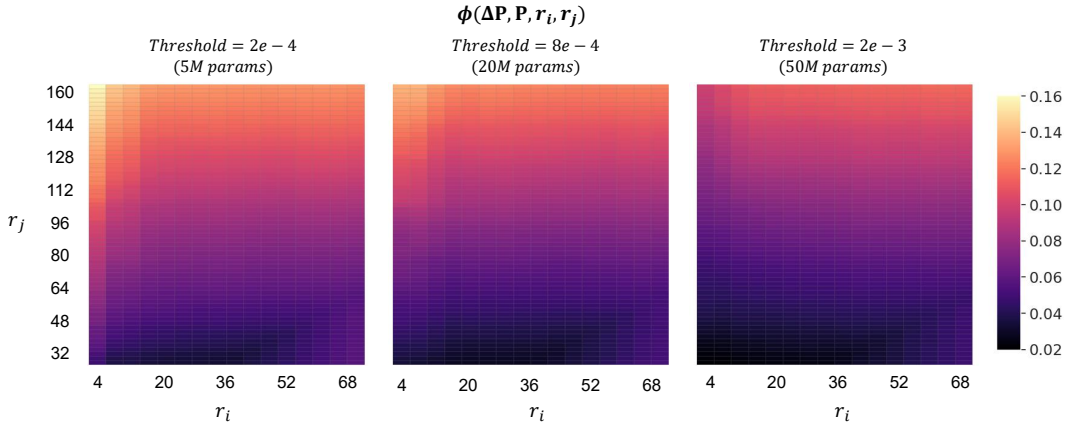

Figure 25: Subspace similarity between ΔP and P under different thresholds.

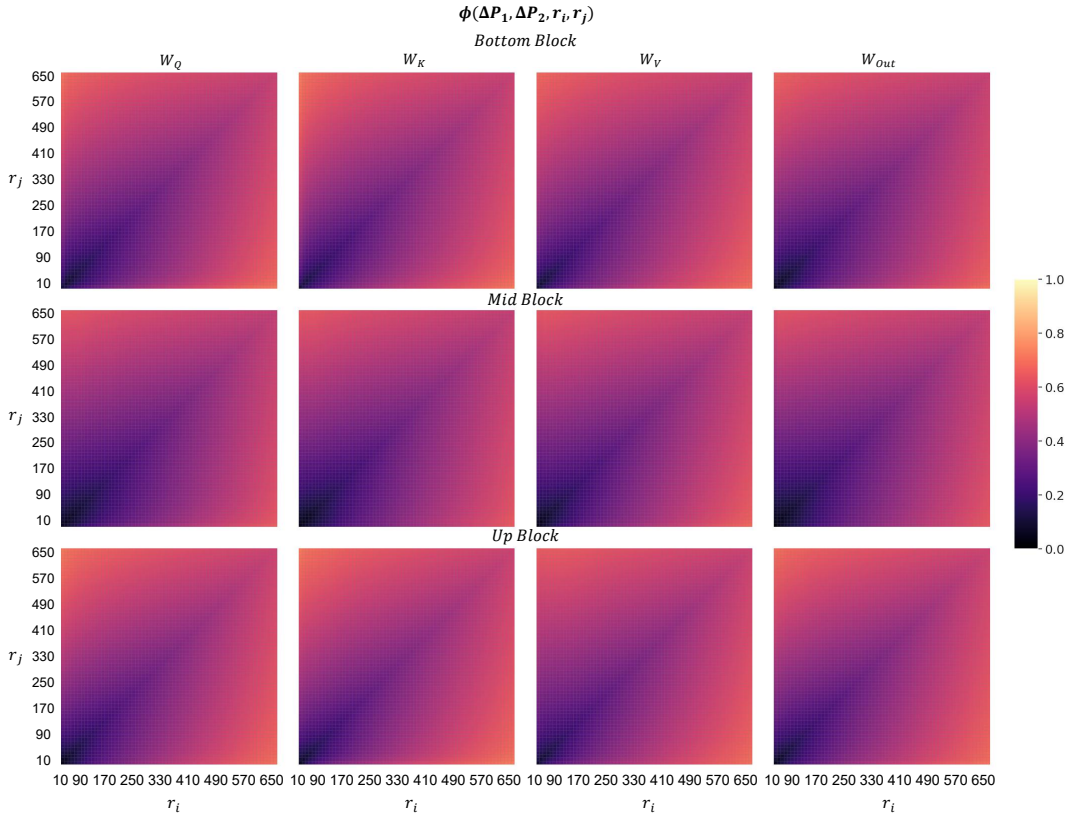

Figure 26: Subspace similarity between ΔP from threshold $\theta_t = 2e-3$ (50M parameters) and $\theta_t = 8e-4$ (20M parameters), in different attention layers, where the x-axis and y-axis represent $\theta_t = 2e-3$ and $\theta_t = 8e-4$ respectively. The subspace similarity across different layers exhibits consistent behavior, demonstrating that the knowledge learned by ΔP remains invariant across layers and modules, indicating strong robustness.

$\theta_t = 8e-4$ respectively., where the heatmaps show almost the same color and distributions, indicating that our conclusion is consistent for the learned matrices from different modules and different attention layers.

Further Analysis of ΔP across Different Threshold Pairs. In the main paper, we analyzed the subspace similarity between the learned matrices ΔP from thresholds of $2e-3$ and $8e-4$, concluding that the learned matrices from different thresholds capture similar task-specific knowledge while emphasizing different directions. In this section, we extend our quantitative analysis to ad-

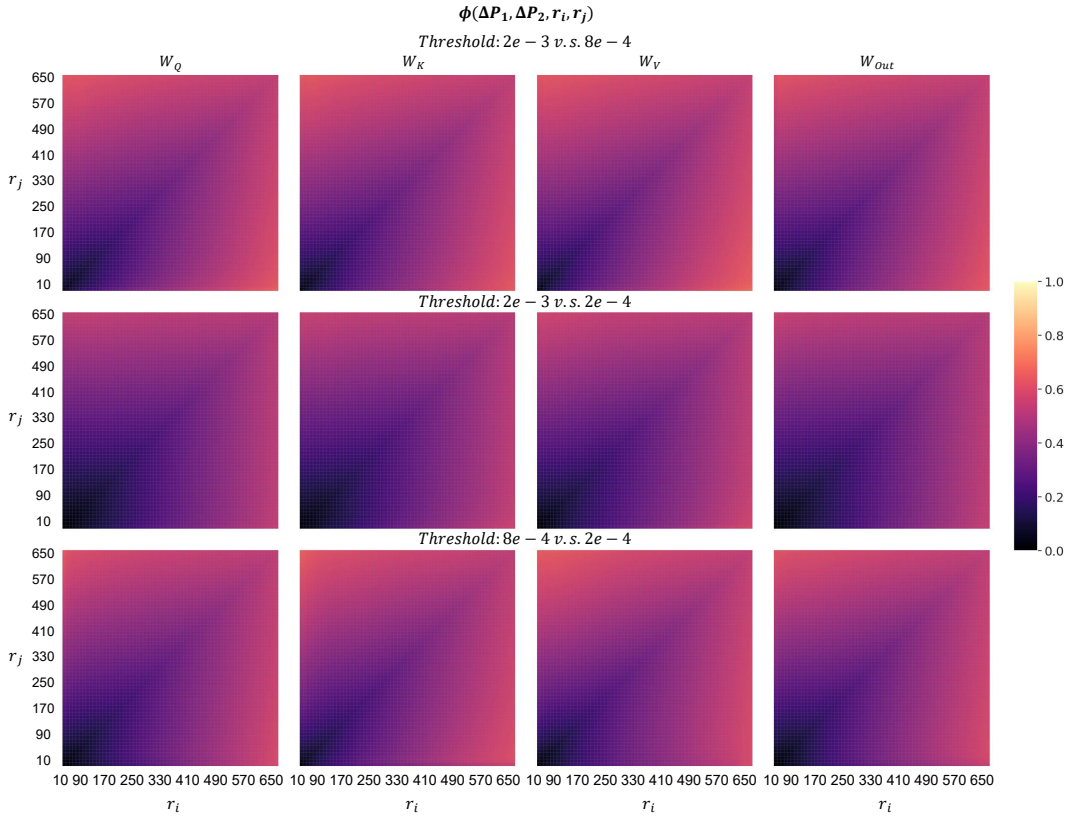

Figure 27: Subspace similarity between ΔP across different threshold pairs: $(2e-3, 8e-4)$, $(2e-3, 2e-4)$, and $(8e-4, 2e-4)$ (from top to bottom).

ditional threshold pairs: $(2e-3, 8e-4)$, $(2e-3, 2e-4)$, and $(8e-4, 2e-4)$. We consider all learnable matrices in the attention module, including the Query, Key, Value, and FFN matrices (corresponding to W_Q, W_K, W_V, and W_{Out}, respectively). The results are presented in Fig. 27. The subspace similarity between the thresholds $(2e-3, 2e-4)$ is lower than that of $(2e-3, 8e-4)$ and $(8e-4, 2e-4)$, suggesting that matrices learned from a closer threshold pair exhibits greater subspace similarity and acquire more similar knowledge.

M LIMITATIONS

Our SaRA focuses on fine-tuning the ineffective parameters of a pre-trained model. However, if the model size is relatively small (e.g., not as large as diffusion models, which typically exceed 100M parameters), the number of ineffective parameters may be insufficient to effectively adapt the model to the downstream dataset. As a result, SaRA is better suited for fine-tuning large models rather than smaller ones. Additionally, since there is no rigorous proof that parameters with the smallest absolute values are always ineffective, caution is warranted to account for potential exceptions, which could lead to reduced performance of SaRA in certain scenarios.