Sparse High Rank Adapters

Kartikeya Bhardwaj[*][§] **Nilesh Prasad Pandey**[*][†] **Sweta Priyadarshi**[†] **Viswanath Ganapathy**[†]
Shreya Kadambi   **Rafael Esteves**   **Shubhankar Borse**   **Paul Whatmough**[§]
Risheek Garrepalli   **Mart Van Baalen**   **Harris Teague**[§]   **Markus Nagel**[§]
Qualcomm AI Research[‡]
[§]{kbhardwa,pwhatmou,hteague,markusn}@qti.qualcomm.com

Abstract

Low Rank Adaptation (LoRA) has gained massive attention in the recent generative AI research. One of the main advantages of LoRA is its ability to be fused with pretrained models, adding no overhead during inference. However, from a mobile deployment standpoint, we can either avoid inference overhead in the fused mode but lose the ability to switch adapters rapidly, or suffer significant (up to 30% higher) inference latency while enabling rapid switching in the unfused mode. LoRA also exhibits concept-loss when multiple adapters are used concurrently. In this paper, we propose Sparse High Rank Adapters (SHiRA), a new paradigm which incurs no inference overhead, enables rapid switching, and significantly reduces concept-loss. Specifically, SHiRA can be trained by directly tuning only 1-2% of the base model weights while leaving others unchanged. This results in a highly sparse adapter which can be switched directly in the fused mode. We further provide theoretical and empirical insights on how high sparsity in SHiRA can aid multi-adapter fusion by reducing concept loss. Our extensive experiments on LVMs and LLMs demonstrate that finetuning only a small fraction of the parameters in the base model significantly outperforms LoRA while enabling both rapid switching and multi-adapter fusion. Finally, we provide a latency- and memory-efficient SHiRA implementation based on Parameter-Efficient Finetuning (PEFT) Library which trains at nearly the same speed as LoRA while consuming up to 16% lower peak GPU memory, thus making SHiRA easy to adopt for practical use cases. To demonstrate rapid switching benefits during inference, we show that loading SHiRA on a base model can be 5×-16× faster than LoRA fusion on a CPU.[¶]

1 Introduction

Low Rank Adaptation (LoRA) [13] is an established technique to tune the behavior of large generative models such as Large Language Models (LLMs) [30, 29] and Stable Diffusion [24, 22]. As the name suggests, LoRA requires very few parameters since it trains low rank projection weights that consume very low memory during the finetuning process while producing excellent results. Moreover, these low rank weights can be fused analytically into the base model, thereby incurring no additional overhead during inference.

Despite its success, there are still several limitations of low rank adaptation methods. First, if LoRA parameters are fused into the corresponding pretrained base model weights, they modify the entire weight tensor. Therefore, deploying LoRA on large models such as LLaMA-1/2 (7B+ parameters) or Stable Diffusion (1.5B+ parameters) on mobile devices would require changing a large number of weights during inference. Consequently, for mobile scenarios, if an application requires *rapid adapter switching*, existing low rank methods would incur a significant memory and latency cost. This is a major deployment challenge because, unlike large GPUs, local memory of small AI accelerators is limited and cannot store all weights at the same time. These challenges can be partially addressed by

---

[*]Equal contribution. [†]Work done while employed at Qualcomm AI Research. [‡]Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc. [¶]Code: https://github.com/Qualcomm-AI-research/SHiRA.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).
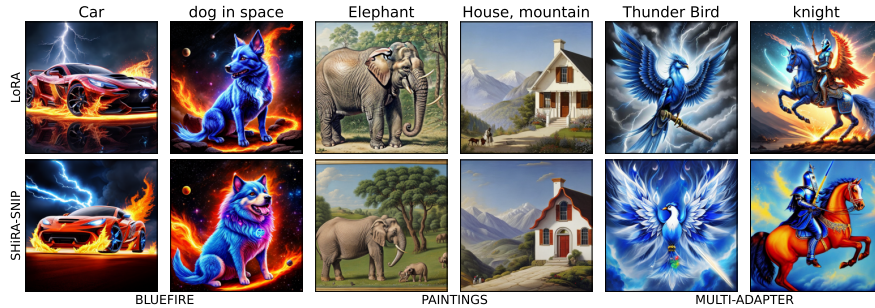
Figure 1: S̲parse H̲igh R̲ank A̲dapters (SHiRA): Changing about 1-2% weights of the pretrained generative model is often sufficient to achieve high performance. Due to its extreme sparsity, SHiRA enables rapid switching and also reduced concept loss during multi-adapter fusion. In contrast, LoRA modifies majority of parameters when fused, thus prohibiting rapid switching on mobile devices, and also experiences concept loss during multi-adapter fusion. For LoRA, elephant for single "paintings" adapter case has artifacts (extra/broken tusks); bird and knight for multi-adapter case lose "paintings" concept and keep only the "blue fire" effects. SHiRA does not experience these issues.

running LoRA in unfused mode; however, unfused inference can incur as high as **30% additional latency** compared to the base model [1] (see section 2.1 for details). This increased inference time in unfused mode and time for adapter switching significantly hampers user experience; hence, this is an important problem which has been a focus of recent research by various industries [9]. Second, LoRA has a well-known limitation called *concept loss* when using multiple concurrent adapters, e.g., combining multiple style transfer adapters, etc. Specifically, it has been well documented [34, 26, 8] that a simple additive merging of multiple LoRA adapters leads to concept loss of one or more adapters. Finally, recent literature also contributes important theoretical and empirical knowledge towards the value of *high rank adapters*. For instance, Kalajdzievski [16] shows that the high rank adapters can greatly outperform low rank adapters when used with correct scaling factors. This calls for further investigation into whether other high rank adapters would significantly outperform LoRA.

In view of the above, we address the following **key problems** in this paper: (*i*) How can we perform rapid switching for fused adapters? (*ii*) Is there a simpler solution for multi-adapter fusion to reduce concept loss? (*iii*) Can we build high rank adapters that have high expressive power without significantly increasing the training or inference costs?

To this end, we propose S̲parse H̲igh R̲ank A̲dapters (SHiRA), a single solution to all three problems above. SHiRA is a highly sparse but a high rank adapter which relies on training only a very small subset of parameters from the original pretrained network. One of the crucial insights we demonstrate is that even finetuning merely 1-2% parameters of the pretrained generative model is sufficient to achieve high performance on many adapter tasks (see Fig. 1). However, unlike LoRA layers that modify all parameters in the weight tensors in the fused mode, SHiRA still keeps a very low percentage of parameters that need to be switched, thus enabling rapid switching at inference time. Moreover, since the pretrained weights are huge, SHiRA being a very sparse adapter greatly aids multi-adapter fusion by significantly reducing concept loss. Finally, we theoretically and emprically analyze the high rank vs. sparsity properties of SHiRA and why that helps with adapter performance.

Overall, we make the following **key contributions**:

- We propose SHiRA, a new high rank adapter paradigm to demonstrate that changing as few as 1-2% parameters of the original network is sufficient for adaptation. Our crucial insight is that even the most basic masking criteria (to identify the top 1-2% parameters) enable SHiRA to significantly outperform LoRA on diverse vision and language tasks.
- SHiRA enables on-device *rapid adapter switching* and provides a natural multi-adapter fusion technique due to high sparsity, thus, significantly reducing *concept loss*. We also theoretically analyze SHiRA through the lens of *high rank adaptation* vs. sparsity.
- We conduct extensive experiments on LLMs (LLaMA-7B, LLaMAv2-7B) and LVMs (Stable Diffusion, SDXL) where we demonstrate that SHiRA significantly outperforms LoRA on both single- and multi-adapter tasks. On LLMs, we show that SHiRA achieves up to 2.7% better accuracy than LoRA on commonsense reasoning. SHiRA also complements advanced variants of LoRA such as DoRA [20] and can be easily applied on top of them.

2

- Finally, on the training side, we provide a PEFT-based latency- and memory-efficient implementation for SHiRA which trains nearly as fast as standard LoRA while consuming 16% lower peak GPU memory. Beyond PEFT, we provide a simple way to turn any trainer into SHiRA finetuning. For inference, we demonstrate that SHiRA weights can be loaded on a CPU up to $5\times$-$16\times$ faster than equivalent LoRA fusing, thereby enabling rapid switching.

The rest of this paper is organized as follows: section 2 presents the background and related work. We propose SHiRA in section 3 while describing its theoretical properties in section 4. We then conduct extensive experiments for SHiRA in section 5. Finally, we discuss the key findings in section 6 and conclude the paper in section 7.

2 Background and Related Work

2.1 Background: Edge Deployment Challenges for LoRA

There are three existing deployment options for LoRA: (*i*) fuse the adapter offline and then deploy on-device: this changes a large fraction of the weight tensors compared to base model which prohibits rapid switching since it will increase DRAM traffic considerably; (*ii*) keep the adapter unfused and run the inference in unfused mode: this can help with rapid switching but would incur significant additional (up to 30% higher) latency as shown in [1] since we would have LoRA branches in the forward pass during inference; (*iii*) use the Huggingface/Diffusers pipeline [1] (built for server-grade GPUs) for mobile inference. This pipeline consists of load→fuse→inference→unfuse→unload to switch adapters. Here, unfused LoRA-A and LoRA-B weights (see Fig. 2(a)) are first loaded into the memory and then fused into the base model by computing $W_{new} = W + AB$; this new weight is used for inference. To switch the adapter, we can unfuse the adapter as $W = W_{new} - AB$ and then unload existing LoRA weights to load the new ones. We provide further evidence in Appendix A to demonstrate that such a pipeline is not feasible for edge devices. This is primarily because edge devices are memory-limited and not all weights of large generative models can be stored in the local memory at the same time. Hence, loading and fusing needs to happen layerwise on a mobile device that obviously results in massive inference latency costs.

2.2 Related Work

LoRA, its variants, and sparse adapters. Many LoRA variants exist in literature: DoRA [20], LoRA+ [11], VeRA [17], LoRA-FA [35], RS-LoRA [16], among many others. The crucial difference between this literature and our work is that we develop a high rank adapter without increasing training and inference costs. Also, for such methods, the final fused adapter still updates all elements in the pretrained weight tensor, thus prohibiting rapid switching. Moreover, for completeness, we will also show that SHiRA is orthogonal to and can be applied on top of some of the latest, more advanced LoRA variants such as DoRA [20] while preserving the benefits of rapid switching.

A few other LoRA variants have also explored a combination of sparsity and low rank adaptation. Examples include RoSA [21], SoRA [6], Sparse-Adapters [12], etc. Among these, Sparse-Adapters [12] explores the use of popular pruning techniques (e.g., SNIP [19]) to prune out adapters to improve their efficiency. SoRA [6] proposes an adaptive rank version of LoRA by gating elements of down and up projection layers and pruning out the zero entries at inference. Finally, RoSA [21] combines a sparse adapter with a low rank one to achieve some high rank benefits. However, since they combine their method with LoRA, the fused adapter weight still overwrites the entire pretrained weight tensor.

Partial Finetuning. Our work is most closely related to partial finetuning techniques that were mostly proposed in the pre-LoRA era [36, 28, 3, 33, 10]. These methods use a mix of fixed sparse masks [28] or learned masks [36, 10] to finetune a pretrained network. Note that, these techniques have been mostly explored for relatively *small* language models, and *not* for recent LLMs and diffusion models. Since the LoRA models exploded in popularity, it has been unclear if other sparse finetuning techniques would achieve comparable results to LoRA on generic adapter tasks, particularly in the vision domain. One *significant* limitation of partial finetuning, as opposed to LoRA-based methods, is its *high GPU memory consumption*, making it *impractical* to be used for large generative models. Consequently, the reduced memory consumption for finetuning was a key factor to LoRA's success and its widespread adoption. To this end, we provide a memory- and latency-efficient PEFT-based implementation for SHiRA which trains as efficiently as LoRA, thus requiring significantly lower memory consumption compared to prior partial finetuning techniques. Further, we explore the effectiveness of sparse finetuning on both large language and vision models and provide a detailed analysis on rapid switching and multi-adapter fusion of the high rank adapters.

3
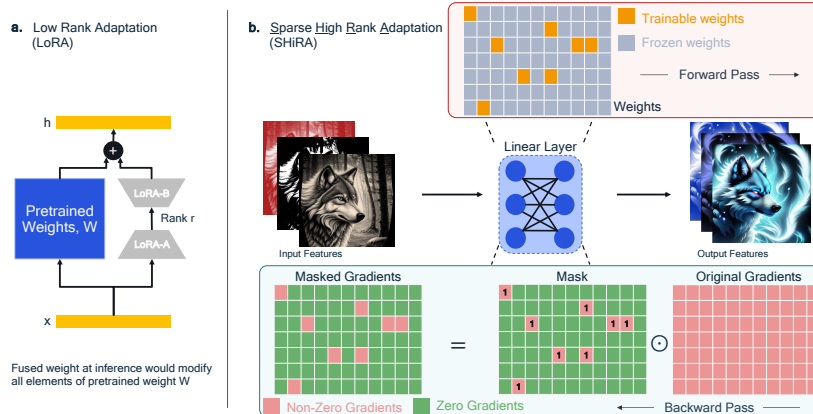
Figure 2: (a) LoRA when fused into the pretrained model modifies all weights and prevents rapid adapter switching. (b) SHiRA does not require additional weights during training but finetunes very few pretrained weights. Our approach relies on a sparse mask for gradient-masking during training. We show that finetuning as low as 1-2% parameters is sufficient to achieve high accuracy.

A notable concurrent work is SpIEL [4] which scales partial finetuning to modern LLMs and also has a PEFT implementation that results in comparable speed and memory as LoRA. The main differences between SpIEL and SHiRA are as follows: (*i*) SpIEL works with dynamic masks while SHiRA uses a static mask. (*ii*) Dynamic mask in SpIEL requires users to install custom sparse linear layer kernels for the GPUs. In contrast, SHiRA does not require installing any custom kernels and directly works with native Pytorch. Hence, SHiRA's biggest advantage is its ease of training/inference deployment. (*iii*) We also analyze multi-adapter fusion properties, e.g., impact of sparsity on orthogonality between adapters, which were not discussed in SpIEL. (*iv*) Finally, SHiRA demonstrates its effectiveness on both vision and language tasks, whereas SpIEL only discusses the language tasks.

Multi-Adapter Fusion. Existing Multi-adapter fusion methods focus on preventing concept loss [8, 34, 26]. However, these methods usually either just use the base LoRA as it is (and then perform some non-trivial postprocessing on them) [34, 26], or some create some minor variants [8]. In contrast, we introduce a new adapter for the concept loss problem where multiple concepts naturally do not interfere with each other. In that respect, our work is orthogonal to the prior multi-adapter fusion work since our adapter can be further postprocessed using such techniques.

3 Proposed Approach

3.1 Sparse High Rank Adapters (SHiRA)

SHiRA exploits highly sparse trainable parameters in the pretrained model. In its simplest form, our adapter can be trained by masking gradients such that only a fraction of original weights get updated. Specifically, we do not add any new weights to the forward pass like LoRA (see Fig. 2(a)) but rather make a small percentage of existing weights trainable (see Fig. 2(b) top). To this end, we first create an extremely sparse (~98-99% zeros) mask $\mathcal{M} \in \mathbb{R}^{n \times m} = \{0, 1\}^{n \times m}$, where n, m are dimensions of the pretrained weight matrix. $\mathcal{M}$ is then used to mask the gradients during backpropagation using a Hadamard product (see Fig. 2(b) bottom). Thus, very few parameters get updated during training and our adapter consists of just those sparse weights. Concrete gradient masking-based and another latency-/memory-efficient PEFT implementations for SHiRA are discussed in section 3.3.

We consider the following masks $\mathcal{M}$ (only 1-2% trainable parameters, see also Appendix B):

1. **SHiRA-Struct:** In this structured mask, certain rows or columns of the weight as well as its diagonal are set to be trainable. All other rows/columns are not trainable. The diagonal makes the mask high rank whereas the structured trainable rows/columns – set to 1 to enable gradient flow to corresponding parameters – lead to a rank 1 adapter. Thus, SHiRA-Struct is a combination of a high rank but very sparse adapter and a rank 1 adapter.
2. **SHiRA-Rand:** This mask is obtained by randomly setting 1-2% parameters as trainable.
3. **SHiRA-WM:** Here we pick top-K parameters to train based on their weight magnitudes (WM), the absolute value of the weight for each layer.

4
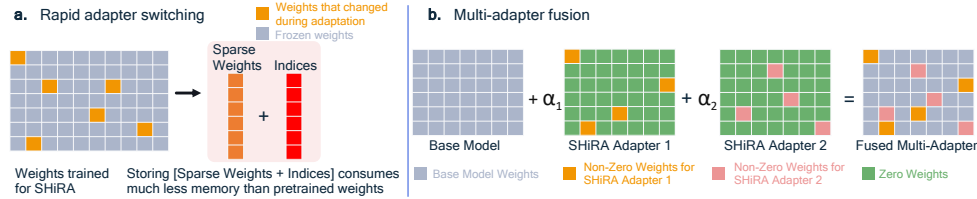
Figure 3: (a) Rapid adapter switching: The sparse finetuned weights can be stored as weights and their indices. At inference time, these weights can be loaded on the base model weights. Since only 1-2% weights need to be overwritten, the adapter can be efficiently switched with different weights at inference, eliminating the need for a separate fusion stage. (b) Multi-adapter fusion: Concept-loss can be reduced if multiple adapters do not significantly interfere with each other.

4. **SHiRA-Grad:** This is a gradient-based mask. We first collect gradients on a small calibration set and then pick top 1-2% weights that receive the highest gradient magnitudes.
5. **SHiRA-SNIP:** The SNIP metric from the pruning literature [19] combines weight magnitude and gradient strategies, i.e., SNIP equals magnitude of the gradient times the weight.

3.2 Rapid Adapter Switching, Multi-Adapter Fusion, and High Rank

Since very few base weights change during the SHiRA training, we can simply extract them out and store them as sparse weights and their indices (see Fig. 3(a)). Hence, SHiRA is comparable to LoRA in model size but overwrites only a fraction of the pretrained weights at inference time. In contrast, LoRA fuses into base weights as $W_{new} = W + AB$ and changes the entire weight. Note that, we do not actually need to fuse SHiRA but rather just need to overwrite the modified value at the correct index in the pretrained weight tensor. This enables rapid switching on resource-constrained devices. To verify that SHiRA indeed provides rapid switching benefits compared to LoRA, we provide an optimized implementation based on `scatter_op` to overwrite base model weights instead of fusing them like LoRA. We demonstrate that on a CPU, **weight loading for SHiRA adapters can be up to 5×-16× faster than equivalent LoRA fusing for inference** (see Appendix C and Fig 7).

Next, we discuss multi-adapter fusion in SHiRA. Given two adapters $\mathcal{A}_1$ and $\mathcal{A}_2$ with sparse masks $\mathcal{M}_1$ and $\mathcal{M}_2$, we ask the following questions: (*i*) What is the impact of sparsity on relative interference between adapters in the multi-adapter setting? (*ii*) Is it possible to create masks that result in nearly orthogonal SHiRA weights so they do not significantly interfere with each other at inference time?

Getting adapters that do not interfere with each other is essential to avoid concept-loss. To this end, we define specific metrics in section 4.2 to analyze orthogonality properties between adapter weights for various SHiRA strategies. We theoretically show that at least one of the SHiRA methods, i.e., SHiRA-Struct can in fact create near-orthogonal adapters. We further experimentally demonstrate in section 5.2.2 that SHiRA-Struct indeed outperforms other methods for multi-adapter fusion.

Finally, since we do not have any low rank weights in the forward pass, our proposed adapters can be high rank albeit highly sparse. We theoretically analyze the rank vs. sparsity properties in section 4.

3.3 Memory- and Latency-Efficient SHiRA Training

We have created two implementations for SHiRA: (*i*) a backward hook-based gradient masking to turn any trainer into SHiRA finetuning (see Appendix D), and (*ii*) a PEFT-based implementation. As discussed in Appendix E, the PEFT-based SHiRA implementation consumes **16.63% lower peak GPU memory and trains almost at a similar speed as LoRA**. On the contrary, DoRA exhibits a 40.99% and 28.9% increase in memory and training time respectively compared to LoRA.

4 Theoretical Insights for SHiRA

4.1 Rank vs. Sparsity

Below we discuss parameter and learning complexity, parallels between LoRA and SHiRA, as well as its optimization properties from the lens of rank and sparsity.

Lemma 4.1. *The parameter complexity and learning complexity of SHiRA is equal to the number of non-zero elements in the adapter.*

Appendix F.1 provides the proof. This lemma suggests that despite high rank property of SHiRA, it would not require significantly larger datasets to converge.

5

Lemma 4.2. *If we specify a sparsity factor, the LoRA is r rank approximation of SHiRA with approximation error bounded by σ_{r+1}^2, the $(r+1)^{th}$ singular value of the SHiRA adapter.*

The above lemma is proved in section F.2. As a consequence of this lemma, any r rank LoRA adapter of size (m,n) can be seen as an approximation of a SHiRA adapter with $mr + rn$ non-zero elements.

Lemma 4.3. *Scaling factor for SHiRA is independent of the rank of the adapter and can be set to 1.*

Please see the proof in Appendix F.3. Lemma 4.3 states that we do not need scaling factors to stabilize the training and, therefore, we do not need additional hyperparameters like α or independent learning rates for separate A and B matrices like in LoRA[13] or LoRA+ [11]. Of note, the scaling factor α can still be used at inference time to vary the intensity of the adapter.

4.2 Adapter Weight Orthogonality in Multi-Adapter Fusion

In this section, we provide theoretical and empirical insights by studying properties of SHiRA and LoRA adapter designs for multi-adapter fusion.

Lemma 4.4. *Consider two adapters, ΔW_1 and ΔW_2. If one of the adapters, ΔW_1 or ΔW_2 lies in the null space of the other, then the adapters will not interfere multiplicatively.*

Proof is given in Appendix F.4. The above lemma implies that two adapters can be efficiently fused without interference if they are orthogonal. In order to analyze the orthogonality between any two adapter weights, we define the following metrics:

Definition 1. *Adapter Weight Orthogonality Magnitude (AWOM)* is defined as the l_2 norm of the product $\mathcal{A}_1^T \mathcal{A}_2$ for two sparse adapter weights $\mathcal{A}_1, \mathcal{A}_2 \in \mathbb{R}^{n \times m}$. AWOM enables us to understand how far the product $\mathcal{A}_1^T \mathcal{A}_2$ is from a zero matrix $\mathbb{O} \in \mathbb{R}^{m \times m}$ ($\mathbb{O}_{i,j} = \{0\} \forall i, j$).

Definition 2. *Adapter Weight Orthogonality Ratio (AWOR)* is defined as the sparsity ratio of the product $\mathcal{A}_1^T \mathcal{A}_2$. Specifically, AWOR $= \left[1 - \left(\frac{\|\mathcal{A}_1^T \mathcal{A}_2\|_0}{m^2} \right) \right]$, where m^2 is #elements in $\mathcal{A}_1^T \mathcal{A}_2$.

Together, AWOM and AWOR can provide us an idea of relative orthogonality between adapter weights $\mathcal{A}_1$ and $\mathcal{A}_2$. Next, we analyze how at least one of the SHiRA strategies (i.e., SHiRA-Struct) can result in near-orthogonal adapters. Recall that, SHiRA-Struct adapters train certain rows/columns and the diagonal elements while keeping all other parameters frozen. Hence, the final trained adapter (after subtracting the pretrained weight) contains a structured pattern of rows/columns and diagonal elements, everything else being zero. Now, without loss of generality, consider two SHiRA-Struct adapters for a layer with square $m \times m$ weights: $\mathcal{A}_1 = \mathbb{I} + \mathcal{S}_1$ and $\mathcal{A}_2 = \mathbb{I} + \mathcal{S}_2$, where $\mathcal{S}_1$ and $\mathcal{S}_2$ are row-wise patterns of trained weights for two different tasks, and $\mathbb{I}$ is an identity matrix. Also, $\mathcal{S}_1$ and $\mathcal{S}_2$ are non-overlapping, e.g., both have same number of non-zero rows but are offset from each other such that they do not have any common trained rows. Then, the following result holds:

Lemma 4.5. *Non-overlapping SHiRA-Struct adapters are nearly orthogonal: AWOR for non-overlapping SHiRA-Struct adapters is at most the sum of sparsity of individual adapters. Since all SHiRA masks are highly sparse, $\mathcal{A}_1^T \mathcal{A}_2$ has a lot of zeros, thus making the adapters nearly orthogonal.*

Proof is provided in Appendix F.5.

We demonstrate the orthogonality properties of various adapters and report the simulation results in Fig. 4. For our experiment, we compute AWOM and AWOR for a variety of adapter designs - dense, sparse-LoRA [12] (sparse LoRA A and B weights), SHiRA-WM and SHiRA-Struct based adapters. As shown in Fig. 4, both dense and sparse LoRA have low AWOR for adapters with larger dimensions, e.g., 4096×4096 which is typical in LLMs. This signifies that these adapter weights are non-orthogonal. On the contrary, SHiRA-WM achieves much higher AWOR than the LoRA variants. More interestingly, SHiRA-Struct is nearly orthogonal. Note that, due to high sparsity, AWOM also tends to be much lower for SHiRA adapters than the dense counterparts. Combined with the fact that AWOR of SHiRA
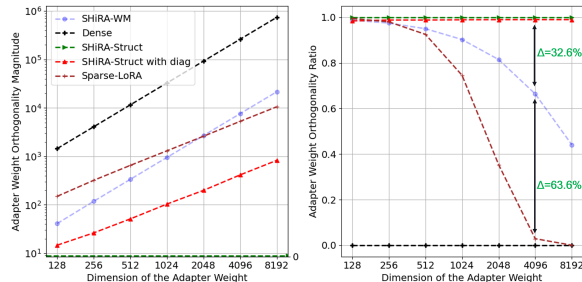
Figure 4: Comparison of average AWOM *(left)* and AWOR *(right)* for 50 randomly initialized adapters. We compare different adapters, namely - Dense, Sparse LoRA, SHiRA-WM and SHiRA-Struct.

6

adapters is 63-96% higher sparsity than LoRA, this may suggest that $\mathcal{A}_1^T \mathcal{A}_2$ would be closer to zero for SHiRA adapters, thus potentially bringing them closer to orthogonality and less interference. Finally, although we have shown interesting properties for SHiRA-Struct, it is still a rank 1 + diagonal adapter. Hence, we need to tradeoff single adapter performance (which strongly depends on adapter's expressive power) against the multi-adapter fusion capabilities. For instance, next we will see that while SHiRA-Struct is good for vision, SHiRA-SNIP performs well across both LVMs and LLMs.

Remark 1. The orthogonality property shown here can lead to disentangled representation for adapter outputs before they merge into the base model. However, this property does not hold for other SHiRA masks that do not have a regular sparsity pattern like SHiRA-Struct even if other SHiRA strategies are still more orthogonal than LoRA weights (e.g., see SHiRA-WM AWOR in Fig. 4(right)). Interestingly, for unstructured sparse masks like SHiRA-WM, SHiRA-Grad, SHiRA-SNIP, etc., both overlapping and non-overlapping adapters have similar orthogonality properties. We discuss this in more detail in section 5.3.2. Finally, this analysis only focuses on *orthogonality of adapter weights* and *not* on orthogonality of subspaces. We leave the subspace analysis of SHiRA for future work.

5 Experiments

5.1 Training Setup and Datasets

For the vision tasks, we use the RealisticVision-v3 model checkpoint for Stable Diffusion-v1.5, and finetune it using different adapters on two style transfer datasets collected using public domain images. The first dataset is called Bluefire which provides a "blue fire" effect to images. The second dataset is a painting dataset which gives a "paintings" effect (see Appendix section G for more details). For both these datasets, we conduct single- and multi-adapter experiments. To quantify the image quality, we use the Human Preference Score-V2 (HPSv2) [32].

On the language domain, we experiment with LLaMA 7B [29], LLaMA2-7B [30] and evaluate it on various commonsense reasoning benchmarks such as HellaSwag, PIQA, SIQA, BoolQ, Arc-easy, Arc-challenge, OpenBookQA and Winogrande. Similar to our vision investigations, we conduct single- and multi-adapter experiments on LLMs as well. Specifically, for language finetuning, we follow the setup adopted by [14, 20] for training and evaluating LoRA [13], DoRA [20], and SHiRA based finetuned models on downstream tasks.

Finally, we also explore generalizability of SHiRA to other popular LoRA models and applications such as SDXL [22] and DreamBooth [25]. Detailed training setups are provided in the Appendix H.

5.2 Vision Results

5.2.1 Impact of Various SHiRA Masks

We first evaluate the image quality for SHiRA and LoRA on Paintings and Bluefire datasets for both single and multi-adapter usecases. Fig. 1 demonstrates comparison between SHiRA-SNIP and LoRA. As evident, by merely changing 2% pretrained weights, SHiRA generates high quality images for both finetuning tasks.

Next, we compare various types of SHiRA masks in Fig. 5. Clearly, all SHiRA schemes produce impressive images for different prompts and significantly outperform LoRA. We further quantify the image quality using HPSv2 for each of the masks. The results are presented in Table 1. As evident, all variants of SHiRA consistently achieve superior or similar HPSv2 scores than LoRA, especially for larger α (see details on scaling factor α in Appendix I). More results are provided in Appendices J and K: see Table 10 and Fig. 10, 11, 12.

Style	Method	%Params	HPSv2 score($\uparrow$)	
			$\alpha = 1$	$\alpha = 0.5$
Paintings	LoRA	3.84	24.7 ± 1.8	31.3 ± 1.5
	SHiRA-Struct	**1.99**	**31.2 ± 1.7**	**33.0 ± 1.8**
	SHiRA-Grad	2.05	30.3 ± 1.8	32.3 ± 1.8
	SHiRA-SNIP	2.05	29.8 ± 1.8	31.6 ± 1.8
Bluefire	LoRA	3.84	32.6 ± 1.9	33.6 ± 1.6
	SHiRA-Struct	**1.99**	**34.2 ± 1.6**	**34.1 ± 1.5**
	SHiRA-Grad	2.05	**34.2 ± 1.5**	33.7 ± 1.7
	SHiRA-SNIP	2.05	33.7 ± 1.7	33.7 ± 1.6

Table 1: HPSv2 score of various adapters on Paintings and Bluefire. SHiRA-Struct outperforms all other methods.

5.2.2 SHiRA Adapters aid Multi-Adapter Fusion

As explained in section 4.2, high sparsity of SHiRA reduces their AWOM and increases the AWOR metrics by increasing the number of zeros in $\mathcal{A}_1^T \mathcal{A}_2$ product even for unstructured schemes such as SHiRA-WM, SHiRA-Grad, and SHiRA-SNIP. We hypothesized that this may lead to improved multi-adapter fusion performance. This was also pointed out by [26, 8, 31]: naively merging multiple LoRA adapters leads to poor performance and concept loss.

7

Figure 5: Comparison between different SHiRA masking methods for single- and multi-adapter image generation. For multi-adapter fusion, SHiRA-Struct outperforms all other adapters by generating exceptional images with high frequency details and good concept fusion (e.g., see fox and flower).

We now validate the effectiveness of various SHiRA schemes on multi-adapter fusion. The right two columns in Fig. 1 and Fig. 5 show our results. SHiRA is clearly better at capturing both concepts than LoRA. For example, both bird and knight images in Fig. 1 generated with LoRA lose most of the paintings concept. Similarly, for the fox image in Fig. 5, LoRA does not show significant bluefire concept. In contrast, SHiRA-Struct and SHiRA-SNIP consistently perform well on many different prompts and produce exceptional images for multi-adapter fusion. Please refer to Appendix K.1 (Fig. 10, 11, 12, and 13) for additional results. For certain classes that were not included in the training set for both adapters (e.g., see Koala in Fig. 10, 12, and 13 in Appendix), we observe that LoRA produces significant artifacts whereas SHiRA generates high quality images.

5.3 Language Results

5.3.1 Single Adapter SHiRA Finetuning

Similar to vision results, we demonstrate the effectiveness of SHiRA on language tasks. For our experiments, each adapter (i.e., weight-magnitude, gradient-magnitude, and SNIP based SHiRA) is trained on the combined 170K sample commonsense reasoning dataset released by [14, 20]. Similar to [20], we train our SHiRA adapters for 3 epochs and compare it against the LoRA baselines. As shown in Table 2, various SHiRA adapters outperform LoRA by 1.9-2.7% on an average on LLaMA-7B. Importantly, SHiRA only modifies 1% base parameter weights as compared to **66.72%** (**4.5B weights**) changed by LoRA in the fused mode, thus enabling rapid switching on edge devices. Interestingly, we found that SHiRA-Struct does not perform well on language tasks likely because it is a rank 1 + diagonal adapter and may not have sufficient expressive power.

Moreover, when compared to newer techniques like DoRA [20], our proposed work takes an orthogonal approach by finetuning very few parameters of the pretrained weights. This strategy allows for an efficient integration of our adapter with methods like DoRA to improve the expressiveness of the adapters. As we show in Table 2, our proposed adapter benefits from DoRA based finetuning and achieves almost comparable performance (within 0.3%) to DoRA on an average, with an added benefit of changing only 1% parameters at inference time. In contrast, DoRA would lead to **66.72%** (**4.5B weights** $\approx$ **9GB memory** in FP16 format) parameter change in the fused mode. Therefore, SHiRA is orthogonal to other existing low rank methods and can be efficiently integrated with them.

8

Model	%Params	%C	BoolQ(↑)	PIQA(↑)	Arc-e(↑)	Arc-c(↑)	WG(↑)	OBQA(↑)	HS(↑)	SIQA(↑)	Avg.(↑)
LoRA	**0.83**	66.72	68.9	80.7	77.8	61.3	78.8	74.8	78.1	77.4	74.7 (+0%)
SHiRA-Grad	1.0	1.0	68.4	80.9	80.2	64.7	**80.4**	78.2	80.3	**79.4**	76.6 (+1.9%)
SHiRA-WM	1.0	1.0	**69.6**	**81.6**	81.5	66.5	79.8	79.4	79.6	77.8	77.0 (+2.3%)
SHiRA-SNIP	1.0	1.0	68.3	80.6	81.5	67.9	80.0	**79.6**	82.1	79.1	**77.4** (+2.7%)
DoRA	0.84	66.72	68.5	**82.9**	81.4	**65.8**	80.8	81.0	84.8	79.6	**78.1** (+0%)
SHiRA-WM-DoRA	6.25*	1.0	70.9	81.9	**81.7**	64.9	**80.8**	79.2	84.5	78.6	77.8 (-0.3%)

Table 2: Evaluation of LLaMA-7B on Commonsense Reasoning. WG and HS denote WinoGrande and HellaSwag, respectively. %C represents parameters changed in the fused mode. (↑): the higher the better. Green denotes improvement. *Trained by masking a high-rank DoRA with a WM mask of top 1% weights, thus changing only 1% of the model during both training and inference.

Model	%Params	%C	BoolQ(↑)	PIQA(↑)	Arc-e(↑)	Arc-c(↑)	WG(↑)	OBQA(↑)	HS(↑)	SIQA(↑)	Avg.(↑)
LoRA	**0.83**	66.72	69.90	79.9	79.8	64.7	82.6	81.0	83.6	**79.5**	77.61 (+0%)
DoRA	0.84	66.72	**71.8**	**83.7**	**83.7**	68.2	**82.6**	**82.4**	89.1	76.0	**79.68** (+2.07%)
SHiRA-SNIP	1.0	1.0	70.42	81.71	83.25	**68.6**	80.51	81.0	**89.78**	79.01	**79.28** (+1.67%)

Table 3: Results for LLaMA2-7B on Commonsense Reasoning.

Finally, we experiment with LLaMA2-7B [30] and demonstrate that SHiRA-SNIP – which achieved the best results on LLaMA-7B – yields significant accuracy gains compared to LoRA and nearly the same accuracy as DoRA (within 0.4%, see Table 3).

5.3.2 Multi-Adapter Fusion on LLMs

We now extend our LLM experiments to the multi-adapter fusion setting. To this end, we create a *new* setup where we independently train multiple adapters on training sets of individual commonsense reasoning benchmarks, i.e., one adapter each for BoolQ, PIQA, and Arc-Easy. In contrast, each adapter in section 5.3.1 was trained on a combined dataset containing 170K samples from all eight commonsense benchmarks as proposed in [14, 20]. In the present section, the goal is to evaluate how much accuracy drop various adapters experience when we perform multi-adapter fusion. Due to its simplicity towards constructing a mask, we will use SHiRA-WM in the rest of this paper. Further, we explore two settings - overlapping and non-overlapping SHiRA-WM adapters. The overlapping mask consists of top 1% parameters being trained for all tasks. On the other hand, the non-overlapping setting trains the top 1% weights for the first task, next top 1% for the second task, and so on. We compare the performance of both LoRA and SHiRA across the multi-adapter fusion of these three tasks. As shown in Table 4, both overlapping and non-overlapping multi-SHiRA outperform multi-LoRA on all three commonsense benchmarks. This is inline with our theoretical analysis in section 4.2 where we suggest that even unstructured sparse SHiRA adapters such as SHiRA-WM would have more orthogonal behavior than LoRA due to high sparsity (see higher AWOR of SHiRA-WM in Fig. 4(right)). In comparison, independently trained LoRA adapters would have no such property and suffer greatly during multi-adapter fusion. As a result, we see that both SHiRA models outperform LoRA by more than 6.5% accuracy on average. Further analysis of the properties of these trained adapters is discussed in Appendix K.3 (see Table 13 and Fig. 9).

Of note, this experiment also demonstrates the value of creating a good mask for single adapter performance: Non-overlapping masks achieve lower single adapter accuracy than the corresponding overlapping masks since they train less important parameters. Hence, creating an optimal mask for SHiRA should be of significant interest to future research.

5.4 Content/Style Personalization: Generalizing SHiRA to SDXL and DreamBooth

Finally, we extend SHiRA to focus on DreamBooth [25] using a much bigger vision model called SDXL [22]. We follow a similar setup as adopted by [2]. Specifically, one content (vase) and two style (wooden sculpture and canvas) datasets with five images each were collected from the DreamBooth dataset [25] and public domains, respectively. These datasets were used to train various content and style adapters. For our experiments, we use SDXL [23] as our base model and train both LoRA and SHiRA adapters with comparable trainable parameters on individual single-concept datasets. During training, prompts containing special identifier tokens like "<CONTENT>" or "<STYLE>" (e.g., <SBU> as content token for vase and <SZN> as style token for wooden sculpture and canvas) are used

9

Model	Single Adapter				Multi-Adapter				
	BoolQ(↑)	PIQA(↑)	Arc_e(↑)	Avg(↑)	BoolQ(↑)	PIQA(↑)	Arc_e(↑)	Avg(↑)	%Drop (↓)
LoRA	**80.52**	79.05	75.67	78.41	77.22	71.27	57.45	67.33 (+0%)	11.08
SHiRA-WM-Overlap	78.07	**79.71**	**77.57**	**78.45**	**77.43**	76.88	67.76	**74.02** (+6.69%)	4.43
SHiRA-WM-Non-Overlap	76.94	**79.71**	75.97	77.54	74.22	**78.4**	**69.15**	73.92 (+6.59%)	**3.62**

Table 4: Multi-adapter fusion evaluation of independently trained SHiRA and LoRA adapters on BoolQ, PIQA, and Arc-Easy. %Drop is calculated as drop in average accuracy for multi-adapter fusion compared to the single adapter average accuracy for each adapter.

Figure 6: LoRA- vs. SHiRA-based DreamBooth on SDXL. Prompts for content/style personalization - *left pair*: "A picture of a dog in `<STYLE:WOODEN-SCULPTURE>` style in a bucket", *center pair*: "A picture of a `<CONTENT:VASE>` with flowers", and *right pair*: "A picture of a sunset in `<STYLE:CANVAS>` style". Here, "`<CONTENT>`" and "`<STYLE>`" are special identifier tokens for content/style.

to finetune the SDXL network for content or style personalization, respectively. During inference, similar prompts are used to generate images from LoRA- or SHiRA-based DreamBooth.

Fig 6 shows DreamBooth generated images for LoRA and SHiRA. Clearly, our proposed adapter produces high quality personalized images of target concept in different scenarios. This highlights the broad applicability of our adapter while still preserving the benefits of rapid adapter switching.

6   Discussion

To summarize our main contributions, we highlight that SHiRA – when used with even the most basic pruning metrics (such as weight- or gradient-magnitude, SNIP, structured masks, etc.) – significantly outperforms LoRA on a variety of large-scale tasks in both large vision and large language domains. For LVM style transfer applications, we found that SHiRA-Struct is the most effective masking technique due to its special orthogonality properties that aid multi-adapter fusion. However, SHiRA-SNIP and SHiRA-Grad are not too far behind and achieve competitive performance as SHiRA-Struct. On the LLM commonsense reasoning side, SHiRA-SNIP is the best strategy out of the masking techniques we have considered in this work. Specifically, SHiRA-Struct did not achieve good results on the more complex commonsense reasoning tasks since it is a combination of a rank-1 + a highly sparse diagonal adapter. SHiRA-Grad on LLMs is about 0.8% worse accuracy than SHiRA-SNIP (76.6% vs. 77.4% average accuracy on commonsense reasoning for LLaMA-1). Therefore, in conclusion, for the applications/fields and the masking techniques considered in this paper, SHiRA-SNIP works well across both language and vision domains. Hence, we recommend that SHiRA-SNIP is one of the strongest candidates that we have considered for sparse finetuning.

7   Conclusion

In this paper, we have proposed SHiRA, a new high rank adapter paradigm to demonstrate that even finetuning merely 1-2% parameters of the pretrained generative models is sufficient to achieve high performance on many adapter tasks. We have demonstrated SHiRA's ability to rapidly switch adapters and to avoid concept loss with support from both theory and experiments. Furthermore, we have shown how specially designed sparse masks can lead to near-orthogonal adapter weights which allows for natural multi-adapter fusion. We have conducted extensive single- and multi-adapter experiments on several vision and language tasks to demonstrate the superiority of SHiRA over LoRA. Our latency- and memory-efficient PEFT-based implementation for training SHiRA runs at nearly the same speed as LoRA while consuming about 16% lower peak GPU memory. Finally, for inference, we have provided a `scatter_op` based method that can load our SHiRA $5\times$-$16\times$ faster than equivalent LoRA fusion on a CPU, thus demonstrating our rapid switching benefits.

10

Acknowledgments

We thank anonymous reviewers for insightful comments and constructive feedback which significantly improved the quality of our work.

References

[1] Goodbye cold boot - how we made LoRA Inference 300% faster. `https://huggingface.co/blog/lora-adapters-dynamic-loading`. Accessed: 2024-05-15.

[2] Sdxl lora for dreambooth. `https://github.com/huggingface/diffusers/blob/main/examples/dreambooth/README_sdxl.md`. Accessed: 2024-05-15.

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

[4] 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.

[5] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning*. Cambridge University Press, 2020.

[6] 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.

[7] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[8] Yuchao Gu, Xintao Wang, Jay Zhangjie Wu, Yujun Shi, Yunpeng Chen, Zihan Fan, Wuyou Xiao, Rui Zhao, Shuning Chang, Weijia Wu, et al. Mix-of-show: Decentralized low-rank adaptation for multi-concept customization of diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024.

[9] Tom Gunter, Zirui Wang, Chong Wang, Ruoming Pang, Andy Narayanan, Aonan Zhang, Bowen Zhang, Chen Chen, Chung-Cheng Chiu, David Qiu, et al. Apple intelligence foundation language models. *arXiv preprint arXiv:2407.21075*, 2024.

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

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

[12] Shwai He, Liang Ding, Daize Dong, Miao Zhang, and Dacheng Tao. Sparseadapter: An easy approach for improving the parameter-efficiency of adapters. *arXiv preprint arXiv:2210.04284*, 2022.

[13] 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.

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

[15] Drew A. Hudson and Christopher D. Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering, 2019.

[16] Damjan Kalajdzievski. A rank stabilization scaling factor for fine-tuning with lora. *arXiv preprint arXiv:2312.03732*, 2023.

[17] Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki Markus Asano. Vera: Vector-based random matrix adaptation. *arXiv preprint arXiv:2310.11454*, 2023.

11

[18] Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. Layer-adaptive sparsity for the magnitude-based pruning. *arXiv preprint arXiv:2010.07611*, 2020.

[19] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[20] 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.

[21] Mahdi Nikdan, Soroush Tabesh, and Dan Alistarh. Rosa: Accurate parameter-efficient fine-tuning via robust adaptation. *arXiv preprint arXiv:2401.04679*, 2024.

[22] 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.

[23] 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.

[24] 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*, pages 10684–10695, 2022.

[25] 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*, pages 22500–22510, 2023.

[26] Viraj Shah, Nataniel Ruiz, Forrester Cole, Erika Lu, Svetlana Lazebnik, Yuanzhen Li, and Varun Jampani. Ziplora: Any subject in any style by effectively merging loras. *arXiv preprint arXiv:2311.13600*, 2023.

[27] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

[28] 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.

[29] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

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

[31] Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. Orthogonal subspace learning for language model continual learning. *arXiv preprint arXiv:2310.14152*, 2023.

[32] Xiaoshi Wu, Yiming Hao, Keqiang Sun, Yixiong Chen, Feng Zhu, Rui Zhao, and Hongsheng Li. Human preference score v2: A solid benchmark for evaluating human preferences of text-to-image synthesis. *arXiv preprint arXiv:2306.09341*, 2023.

[33] Runxin Xu, Fuli Luo, Zhiyuan Zhang, Chuanqi Tan, Baobao Chang, Songfang Huang, and Fei Huang. Raise a child in large language model: Towards effective and generalizable fine-tuning. *arXiv preprint arXiv:2109.05687*, 2021.

[34] Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. *arXiv preprint arXiv:2311.03099*, 2023.

12

[35] Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*, 2023.

[36] Mengjie Zhao, Tao Lin, Fei Mi, Martin Jaggi, and Hinrich Schütze. Masking as an efficient alternative to finetuning for pretrained language models. *arXiv preprint arXiv:2004.12406*, 2020.

13

A Edge Deployment Challenges for LoRA (Contd.)

To understand the overhead of each of the stages to the standard huggingface LoRA inference pipeline (i.e., `load`, `fuse`, `unfuse`, `unload`), we experiment with the pipeline provided in [1] and iteratively add adapters to SDXL model [22]. As evident from Table 5, on a server-grade GPU, `load` time dominates whereas `fuse`/`unfuse`/`unload` times are relatively negligible. However, if we try to run the exact same pipeline on an everyday device like a desktop-grade CPU, we see that the `fuse` and `unfuse` times start dominating and can hinder rapid adapter switching. Note that, on an even more constrained device like a mobile phone, AI accelerators do not have sufficient memory to store weights from all layers at the same time in the local memory. Hence, on such devices, we would need to load base model weights for each layer into the local memory, and then fuse corresponding LoRA weights before we can run inference for that layer. This obviously leads to a massive inference latency overhead. As a result, existing deployment options are *not* feasible for rapid switching on mobile devices.

Stage	Server-GPU (s)	Desktop-CPU (s)
load	0.883 ± 0.085	0.786 ± 0.056
fuse	0.306 ± 0.044	3.003 ± 0.023
unfuse	0.206 ± 0.041	2.916 ± 0.014
unload	0.007 ± 0.001	0.007 ± 0.001

Table 5: Latency (in s) to `load`, `fuse`, `unfuse`, `unload` [1] adapters on SDXL on Server-GPU and Desktop-CPU. On a mobile device, fusing/unfusing would happen for each layer iteratively since we cannot store all weights at the same time on local on-chip memory (unlike a large GPU), resulting in much higher overhead.

B More Details on SHiRA Masks

Selecting important salient weights pertinent to a task can be done in many ways, and one popular approach is to use masks. In this section we discuss various strategies to construct sparse mask based on different heuristics to select weights for efficient finetuning of large generative models.

B.1 Structured Sparse Mask (SHiRA-Struct)

This is a simple structured mask. We begin with making every f rows or columns in a weight matrix trainable, where we call f as the *frequency* parameter and we choose it based on how much sparsity we need in the adapter. That is, the mask $\mathcal{M}$ consists of every f rows or columns containing ones and everything else as zeros. This actually makes it a rank 1 mask because all rows and columns would be linearly dependent. Therefore, to make it high rank, we also add a *diagonal* parameter which makes the resulting mask $\mathcal{M}$ high rank.

B.2 Unstructured Sparse Random Mask (SHiRA-Random)

Unstructured sparse random masks involve masking individual weights without any specific pattern or structure. The masked weights are randomly scattered throughout the weight tensor, resulting in a sparse weight tensor. However, as the weights are selected without considering their salience to the task, randomly selected unstructured masks may often be sub-optimal for finetuning. One common way of constructing random sparse marks is using Bernoulli sampling:

$$f(k;p) = \begin{cases} p & \text{if } k = 1, \\ 1 - p & \text{if } k = 0. \end{cases} \tag{1}$$

where, p is the probability of sampling 1 from the distribution.

B.3 Weight Magnitude-Based Sparse Mask (SHiRA-WM)

Many earlier works [18, 27] have shown the importance of weight magnitude based masks for identifying important weights in the network. Motivated by this literature, we design a weight magnitude based proxy to adapt the behavior of the pretrained network. Specifically, we create a mask by choosing the top-K weight magnitudes at specific layers where SHiRA is employed. We finetune only these top-K weights and keep the rest of them frozen to their pretrained values. Typically, K is a very small percentage of parameters so that the overall number of parameters to be tuned stays comparable to LoRA and its variants.

14

B.4 Gradient Based Sparse Mask (SHiRA-Grad)

Despite the efficacy of employing weight magnitude based scheme, this approach lacks an inherent awareness of the specific task for which the model is being finetuned. To address this challenge, we design a similar gradient magnitude based proxy to identity important top-K weights for the task and only adapt them during the finetuning process.

B.5 SNIP Based Sparse Mask (SHiRA-SNIP)

SNIP [19] combines both weight and gradient based schemes and is computed as the magnitude of the product of the weight and its corresponding gradient. This formulation effectively captures the interplay between the weight magnitude, which reflects its overall contribution to the model's output, and its gradient information, which encodes the weight's task-specific relevance during finetuning.

SNIP for a weight parameter is defined as:

$$SNIP \triangleq |\langle \Theta_i, \nabla_{\theta_i} \mathcal{L} \rangle| \tag{2}$$

where $\langle . \rangle$ represents inner product, Θ_i is the weight parameter, $\nabla_{\theta_i} \mathcal{L}$ is the gradient of weight parameter with respect to the task loss $\mathcal{L}$ for the ith parameter in the network.

C Fuse and Scatter Op implementation

In this section, we compare fusing times of LoRA with our efficient `scatter_op` (`torch.Tensor.scatter_`) based implementation for SHiRA. For our experiments, we perform benchmarking on a Desktop-grade CPU and compute the average times for various tensor dimensions (e.g., tensor dimension = 4096 implies a weight of size 4096×4096, which is typical in modern LLMs). As shown in Fig. 7, our `scatter_op`-based SHiRA inference pipeline is up to $13\times$-$16\times$ faster than fusing LoRA weights, specially for larger dimensions.
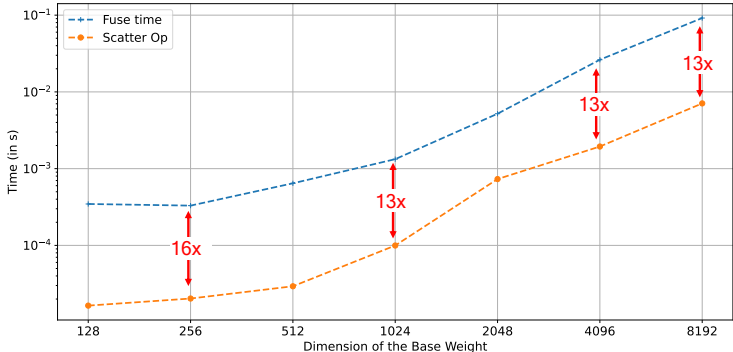
Figure 7: Comparison between average times for LoRA-fuse and SHiRA-`scatter_op` implementation for 50 randomly initialized weights of various dimensions on a CPU (e.g., dimension = 4096 means that the weight has shape 4096×4096). For fusing, we compute time taken to merge LoRA adapters into the base weights (W + AB). Similarly, for the `scatter_op`, we report time taken to overwrite base weights with SHiRA weights using the scatter op (`torch.Tensor.scatter_`) based implementation in Pytorch.

Next, we present end-to-end switching times for prevalent LVMs and LLMs: SDXL and LLaMA2-7B. Notably, even for a smaller model like SDXL (2.6B params compared to 7B params in LLaMA2-7B), SHiRA achieves a 4.68x faster switching time (0.77s vs. 3.6s), while for LLaMA2-7B, with larger tensor dimensions, SHiRA attains a 5.71x speedup (4.93s vs. 28.15s) on a consumer grade CPU (see Table 6). Note that, fusing LoRA adapters for LLaMA2-7B on a CPU is 28.15s (nearly half a minute). Indeed, waiting half a minute for the adapter to switch/fuse is quite substantial and hampers user experience significantly. In contrast, SHiRA can get the adapter ready for inference within 4.93s,

15

Model	LoRA	SHiRA	Speed-up
SDXL	3.64 ± 0.10	$\mathbf{0.77 \pm 0.09}$	$4.68\times$
LLaMA2-7B	28.15 ± 1.62	$\mathbf{4.93 \pm 0.23}$	$5.71\times$

Table 6: End-to-End switching time on CPU for SDXL and LLaMA2-7B: We achieve a very high ($4.7\times$-$5.7\times$) speed up in switching time compared to LoRA.

thus significantly improving the user experience. Note that, once the adapters are fused, inference time on the hardware is equal for both LoRA and SHiRA. Moreover, as discussed in [1], for unfused LoRA case (which can enable rapid switching), the inference latency can be up to 30% higher which is not the case with SHiRA.

D   Turn any Trainer into SHiRA: Gradient Hook based Implementation

In this section, we provide a method to convert any floating point training into SHiRA based finetuning. Specifically, SHiRA can be implemented directly using a functionality called `post_accumulate_gradient_hooks` available in Pytorch 2.1.0. This `gradient_hook` can be used to mask gradients after the gradient accumulation step is completed. Moreover, this enables us to apply SHiRA on any publicly available trainer (e.g., `Transformers.Trainer`, `SFT_Trainer`, etc.). Therefore, implementing SHiRA on any task is trivial and can be done even without PEFT library, thus making SHiRA very easy to implement.

With this gradient hook based implementation, we were able to train all our adapters (including for models such as LLaMA-7B, LLaMA2-7B and SD-1.5) on a single NVIDIA A100 GPU at nearly the same speed as PEFT based LoRA implementation. SHiRA runs at 2.17 it/sec as compared to LoRA which is at 2.42 it/sec for LLaMA-7B finetuning.

E   Latency- and Memory-Efficient PEFT based Implementation for SHiRA

As discussed in Appendix C, `scatter_op` can be utilized to manage sparse weight updates during inference. Given that SHiRA only finetunes a small subset of the pretrained model weights, we adopt a similar `scatter_op`-based approach for training. This allows us to retain only the sparse training parameters in the optimizer, thereby significantly reducing the peak GPU memory utilization during training. As shown in Table 7, SHiRA not only trains at almost similar speed as LoRA, but also consumes $\sim 16\%$ lower peak GPU memory. Compared to other variants like DoRA, SHiRA training consumes significantly ($\sim 40\%$) lower peak GPU memory and also trains much faster (SHiRA is about 36% faster than DoRA). All memory requirement data was collected using `psutil` utility used within the `Transformers.Trainer` training loop for LLaMA2-7B.

Finally, note that, partial finetuning techniques proposed in the pre-LoRA era [36, 28, 3, 33, 10] do not have such memory-efficient implementations, which makes them impractical for large generative models. Therefore, SHiRA significantly outperforms prior partial finetuning techniques in training memory costs and is highly practical for modern LVM and LLM adaptations tasks.

Adapter	Peak GPU memory (GB)	#Training steps/s
LoRA-PEFT	35.10	0.69
DoRA-PEFT	49.49 (+40.99%)	0.49 (-28.98%)
SHiRA-PEFT	29.26 (-16.63%)	0.67 (-2.89%)

Table 7: Peak GPU memory consumption (in GBs) and #Training steps per second during training for PEFT-based implementation of various adapters for LLaMA2-7B. Relative changes compared to LoRA are highlighted: Green indicates improved performance (lower memory consumption, faster training speed), while Red indicates degraded performance (higher memory consumption, slower training speed). SHiRA trains at nearly the same speed as LoRA but consumes up to 16% lower peak GPU memory.

16

F Proofs of Lemma

F.1 Lemma 4.1

Lemma 4.1. *The parameter complexity and learning complexity of SHiRA is equal to the number of non-zero elements in the adapter.*

Proof. The parameter complexity and learning complexity depends on the parameters to be learned. The number parameters of the adapter is equal to $|| \Delta \mathbf{W} ||_0$. □

F.2 Lemma 4.2

Lemma 4.2. *If we specify a sparsity factor, the LoRA is r rank approximation of SHiRA with approximation error bounded by σ_{r+1}^2, the $(r+1)^{th}$ singular value of the SHiRA adapter.*

Proof. Let $\Delta \mathbf{W}$ be the given SHiRA adapter of size (m, n) and sparsity factor ρ. Consider the SVD decomposition of $\Delta \mathbf{W}$. Next, we construct an r rank matrix approximation using the r largest singular values of the adapter. This reconstructed r rank matrix can be seen as a LoRA adapter. Based on Eckart-Young theorem ([7]) and theorem 4.95 in [5], the approximation error is equal to $(r+1)$-th singular value of the SHiRA adapter (σ_{r+1}^2). If the $\Delta \mathbf{W}$ is an r rank matrix then the approximation error is zero. □

F.3 Lemma 4.3

Lemma 4.3. *Scaling factor for SHiRA is independent of the rank of the adapter and can be set to 1.*

Proof. The LoRA update equation for any given adapter is as follows:

$$\mathbf{Y_{out}} = (\mathbf{W} + \alpha_r \mathbf{BA})\mathbf{X_{in}} + \mathbf{b}. \tag{3}$$

Note $\alpha_r = \frac{\alpha}{r}$ is the scaling factor, where α is a hyperparameter and r is the rank. Three possible initialization for $\mathbf{A}$ and $\mathbf{B}$ are as follows:

- if A and B are initialized to zero, no learning occurs since this corresponds to saddle point [11].

- A and B are initialized to $\mathcal{N}(0, \sigma_a^2)$ and 0 respectively. Here, $\sigma_a^2 = \Theta(n^{-1})$, to ensure that $A^T x_i$ remains bounded with width n of the adapter.

- A and B are initialized to 0 and $\mathcal{N}(0, 1)$ respectively. Here, it is important to note that the variance of B does not depend of the width of the adapter.

However, to avoid gradient collapse for higher ranks, [16] recommends to set α_r as $\frac{\alpha}{\sqrt{r}}$. Further, optimal convergence the update of A and B matrix updates have different learning rates [11]. For the SHiRA adapter, the update equation is given below:

$$\mathbf{Y_{out}} = (\mathbf{W} + \mathbf{S})\mathbf{X_{in}} + \mathbf{b}. \tag{4}$$

where, S is the sparse matrix with a designed sparsity ratio. All non-zero locations in S are *implicitly* initialized to the base matrix weights. This initialization ensures that the updates remain bounded during the finetuning stage using stochastic gradient descent. It is also important to note that the scaling is independent of the rank for SHiRA. □

F.4 Lemma 4.4

Lemma 4.4. *Consider two adapters, ΔW_1 and ΔW_2. If one of the adapters, ΔW_1 or ΔW_2 lies in the null space of the other, then the adapters will not interfere multiplicatively.*

17

Proof. The proof leverages two facts: (*i*) $\Delta W_1{}^T \Delta W_2 = \mathbb{O}$ given that one adapter lies in the null space of other. Here, $\mathbb{O}$ is a zero matrix ($\mathbb{O}_{i,j} = \{0\} \forall i,j$). (*ii*) Power series expansion of the non-linear activation function: The power series expansion has terms involving the matrix product of adapters. Since each adapter is in the null space of the other, all terms involving product of adapters are equal to zero. Therefore the adapters do not interfere multiplicatively. □

This lemma can be extended to a scenario with more than two parallel additive adapters. If all possible pairs of adapters lie in the null space of each others all cross-terms between adapters are zero.

F.5 Lemma 4.5

Lemma 4.5. *Non-overlapping SHiRA-Struct adapters are nearly orthogonal. That is, AWOR for non-overlapping SHiRA-Struct adapters is at most the sum of sparsity of individual adapters. Since all SHiRA masks are highly sparse, this means that the product $\mathcal{A}_1^T \mathcal{A}_2$ has a lot of zeros, thus making the adapters nearly orthogonal.*

Proof. Continuing from the adapter definitions used in the main text for this lemma, let us compute $\mathcal{A}_1^T \mathcal{A}_2$ and then analyze its AWOR:

$$\mathcal{A}_1^T \mathcal{A}_2 = (\mathbb{I} + \mathcal{S}_1)^T (\mathbb{I} + \mathcal{S}_2) = \mathbb{I} + \mathbb{I}\mathcal{S}_2 + \mathcal{S}_1^T \mathbb{I} + \mathcal{S}_1^T \mathcal{S}_2 = \mathbb{I} + \mathcal{S}_2 + \mathcal{S}_1^T \tag{5}$$

Here, $\mathcal{S}_1^T \mathcal{S}_2$ is zero by design because $\mathcal{S}_1$ and $\mathcal{S}_2$ do not have common non-zero rows. Moreover, since both $\mathcal{S}_1$ and $\mathcal{S}_2$ are highly sparse, $\mathcal{A}_1^T \mathcal{A}_2$ has a sparsity equal to the sum of sparsity of $\mathbb{I}$, $\mathcal{S}_1$ and $\mathcal{S}_2$. Note that, $\mathbb{I} + \mathcal{S}_2 = \mathcal{A}_2$. Thus, AWOR for non-overlapping SHiRA-Struct adapters is at most the sum of sparsity of individual adapters. □

G   Dataset and Evaluation Metric Descriptions

G.1   Datasets

G.1.1   Language Datasets

For language finetuning tasks, we use the commonsense reasoning datasets, which comprise 8 sub-tasks, each with a pre-defined training and testing set as shown in Table 8. We follow the setting of [14] for SHiRA Single Adapter training. The common sense reasoning training dataset is a combination of the training datasets provided by [15], while we evaluate each evaluation dataset separately as in Table 2. For multi-adapter LLM experiments, we train each adapter from one particular task, and then perform multi-adapter evaluation on all the tasks.

Dataset	#Train	#Val	Test
PiQA	16K	2K	3K
BoolQ	9.4K	2.4K	2.4K
SIQA	33.4K	1.9K	1.9K
OBQA	4.9K	0.5K	0.5K
Winogrande	9.2K	1.3K	1.8K
HellaSwag	39.9K	10K	10K
Arc_easy	2.25K	570	2.36K
Arc_challenge	1.12K	299	1.12K

Table 8: Commonsense Benchmarks

G.1.2   Vision Datasets

For style transfer adaptation tasks as described in sections 5.2.1 and 5.2.2, we use two datasets, Bluefire and Paintings. Images present in both of these datasets are collected from public-domain (CC-0 license).

The Bluefire dataset consists of a total of 54 images consisting of 6 different concepts - Cars, Dragons, Birds, Foxes, Men and Castles. For all these concepts, images with "blue-fire" effect are collected and used for style transfer finetuning. The validation of the Bluefire dataset consists of 30 images. 9 of the 30 images contain one of the 6 concepts in the training set, and the rest 21 are new. A few examples of unseen concepts in the validation set: *football, monster, sword, chess rook, lion, koala etc*.

Similarly, the painting datasets contain a total of 90 images of "painting" style images of 9 different concepts - fire, birds, elephants, ships, horses, flowers, women, men and tigers. The validation set of the Paintings dataset consists of 21 images, out of which 9 contain concepts from the training set. The remaining 12 are new concepts not included in the training set. A few examples of unseen concepts in the validation set: *lion, tiger, dog, cat, koala, panda, and other landscapes*.

18

blazing fiery car, lightning

Alpha=0.0   Alpha=0.25   Alpha=0.50   Alpha=0.75   Alpha=1.0   Alpha=1.25

Figure 8: Effect of α scaling on image quality. $\alpha = 0.0$ is the base model output without any adapter effects. We can see that as the α increases, the SHiRA adapter effect increases similar to how it works for LoRA inference.

G.2 Evaluation Metrics

HPSv2 metric evaluation For all style transfer finetuning experiments with Bluefire and Paintings dataset, we report HPS metric to quantify the quality of the generated images. For Bluefire validation, 30 images per validation prompt are generated for different seeds, hence generating 900 images for HPS analysis. We follow a similar paradigm for Paintings and generate 630 images with 21 prompts.

H Training Details

In this section, we list hyperparameters used for our experiments for Language and Vision finetuning tasks in Table 9.

Method	Adapter Target Modules	Optimizer	LR	LR-Scheduler	Rank
LoRA LVM			$1e-4$	Cosine	64
SHiRA LVM			$1e-4$	Cosine	NA
LoRA LLM	q-proj,k-proj,v-proj,up-proj,down-proj	AdamW	$2e-4$	Linear	32
DoRA LLM			$2e-4$	Linear	32
SHiRA LLM			$5e-4$	Linear	NA

Table 9: Training hyperparameters used for finetuning experiments.

All finetuning and evaluation experiments for language and vision tasks are done using a single NVIDIA A100 GPU.

I Effect of Scaling Factor α during Inference

As described in section 3.1, in order to adapt the pretrained model to a new task, we only finetune very few weight parameters relevant to the task. For our adapter, we can easily extract out these modified weights as $S = W_{new} - W$, where W_{new} is the weight obtained after SHiRA training, and W is the prertained weight. Since only 1-2% weights change during SHiRA training, S is highly sparse and thus constitutes our sparse adapter. Hence, the new finetuned weights of the base model can be viewed as $W_{new} = W + S$.

Similar to LoRA, the strength of SHiRA adapter at inference time can be modified using a scaling factor α. For any defined α scaling, the new weights of the model can be expressed as $W_{new} = W + \alpha S$. Fig. 8 shows the effect of varying α on the output image. As evident, choosing an $\alpha < 1$ reduces the "blue fire" in the generated image and whereas $\alpha > 1$ amplifies the style transfer effect. For $\alpha = 0.0$, the adapter is disabled and the model's output is the same as that for the base model.

J More Detailed Comparison among Various Masks

We provide HPSv2 scores for all SHiRA masking schemes in Table 10.

19

Adapter Style	Adapter Method	%Params	HPSv2 score($\uparrow$)		
			$\alpha = 1$	$\alpha = 0.75$	$\alpha = 0.5$
Paintings	LoRA	3.84	24.7 ± 1.8	28.4 ± 1.4	31.3 ± 1.5
	SHiRA-Struct	**1.99**	$\mathbf{31.2 \pm 1.7}$	$\mathbf{32.1 \pm 1.8}$	$\mathbf{33.0 \pm 1.8}$
	SHiRA-Rand	2.05	30.7 ± 1.9	31.7 ± 1.8	32.7 ± 1.9
	SHiRA-WM	2.05	29.7 ± 1.9	30.6 ± 1.7	32.1 ± 1.8
	SHiRA-Grad	2.05	30.3 ± 1.8	31.3 ± 1.7	32.3 ± 1.8
	SHiRA-SNIP	2.05	29.8 ± 1.8	30.8 ± 1.8	31.6 ± 1.8
Bluefire	LoRA	3.84	32.6 ± 1.9	34.1 ± 1.5	33.6 ± 1.6
	SHiRA-Struct	**1.99**	$\mathbf{34.2 \pm 1.6}$	$\mathbf{34.7 \pm 1.5}$	$\mathbf{34.1 \pm 1.5}$
	SHiRA-Rand	2.05	33.4 ± 1.9	34.1 ± 1.5	33.7 ± 1.7
	SHiRA-WM	2.05	31.9 ± 2.1	33.3 ± 1.6	33.1 ± 1.7
	SHiRA-Grad	2.05	$\mathbf{34.2 \pm 1.5}$	34.4 ± 1.5	33.7 ± 1.7
	SHiRA-SNIP	2.05	33.7 ± 1.7	34.3 ± 1.4	33.7 ± 1.6

Table 10: Comparison between LoRA and various SHiRA schemes with respect to HPSv2 metric. For vision problems, SHiRA-Struct outperforms all other methods.

Adapter	cifar10	cifar100	food101	dtd
LoRA	97.94	87.97	84.27	69.41
SHiRA	**98.05**	**88.15**	**84.43**	**69.73**

Table 11: LoRA vs SHiRA for Image Classification using ViT-Base model. SHiRA consistently outperforms LoRA on these transfer learning tasks.

K More Results

K.1 Additional Sample Images for Vision Style Transfer Applications

We show many more sample images for various adaptation usecases in Fig. 10, 11, 12, and 13.

K.2 Image Classification and GLUE

We further conduct more experiments on image classification and GLUE tasks using SHiRA-WM. For image classification, we finetune Vision Transformer (ViT) using LoRA and SHiRA for four common transfer learning datasets, namely, CIFAR-10, CIFAR-100, Food101, and Describable Textures Dataset (DTD) (see Table 11). Both methods have comparable parameters around 300K. As shown in Table 11, we outperform LoRA on all image classification tasks.

For GLUE, we use the code released by SoRA [6] which relies on dynamically adjusting the ranks of the adapters. In Table 12, we report accuracy on four common GLUE tasks: QNLI, COLA, SST2, and MRPC. Accuracy numbers for LoRA and SoRA are directly taken from the SoRA paper since we are using the official code to run SHiRA experiments. As evident, with nearly 2x smaller adapter, SHiRA outperforms LoRA by 1.1% accuracy on average. Further, SHiRA achieves a similar accuracy as SoRA while being 30% smaller in adapter size. Indeed, SoRA cannot enable rapid switching like SHiRA. Therefore, we again demonstrate that a simple approach like SHiRA-WM outperforms LoRA and its advanced variants with a similar or significantly better accuracy while providing additional deployment benefits.

K.3 Analysis of Trained Adapters

Are adapter tasks sufficiently different? Table 13 shows the L2 analysis for the adapters trained in Table 4. We compute the L2 distance between each adapter and the original pretrained weights (all adapters train top 1% weights in the overlap setting) as well as the L2 distance between each adapter. Clearly, each adapter is closer to the pretrained weights compared to the other adapters. This demonstrates that the tasks are sufficiently different.

Why does SHiRA-WM-Overlap perform well? Next, as shown in Fig. 9, for unstructured SHiRA masks, both overlapping and non-overlapping adapters have identical AWOR and AWOM values. This suggests that their orthogonality characteristics are quite similar due to the high sparsity. We hypothesize that this is the main reason for the good performance of SHiRA-WM-overlap and explains the results in Table 4.

20

Adapter	#Params	COLA	QNLI	MPRC	SST2	Average
LoRA	1.33M	69.73	93.76	89.71	95.57	87.19 (+0%)
SoRA	910K	**71.48**	**94.28**	91.98	95.64	**88.34** (+1.15%)
SHiRA	**636K**	70.62	93.90	**92.15**	**96.50**	**88.29** (+1.10%)

Table 12: GLUE benchmarking for the DeBERTa-V3-base. As evident, with nearly 2x smaller adapter, SHiRA outperforms LoRA by 1.1% accuracy on average. Further, SHiRA achieves a similar accuracy as SoRA while being 30% smaller in adapter size. Hence, SHiRA generalizes to other language tasks as well.

	Base	Arc_e	BoolQ	PIQA
Base	0	**37.0**	**67.0**	**75.0**
Arc_e		0	75.0	81.5
BoolQ			0	98.5
PIQA				0

Table 13: L2 distances between pretrained base weights and SHiRA adapters vs. distances between adapters: Adapters are closer to the base model weights than to each other.
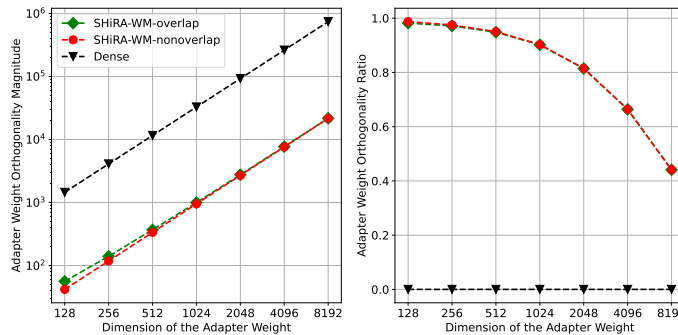
Figure 9: Adapter Weight Orthogonality Magnitude (AWOM: L2 magnitude) and Adapter Weight Orthogonality Ratio (AWOR: Sparsity Ratio) of the product $\mathcal{A}_1^T \mathcal{A}_2$ between two adapters for unstructured SHiRA-WM overlap and non-overlapping cases (99% sparse). We vary the adapter dimensions (e.g., 4096 refers to a pretrained weight of dimensions 4096×4096) and measure AWOM and AWOR for each weight size (averaged over 50 seeds). For unstructured SHiRA masks, overlapping and non-overlapping adapters achieve *coinciding* AWOR and AWOM, thus suggesting that their orthogonality properties are very similar due to high sparsity. This explains our multi-adapter LLM results in Table 4.

L   Societal Impact

Our work enables on-device deployment of adapters which can have a clear positive impact on society as it allows for privacy-preserving generative AI use. With our work, users would be able to rapidly generate images in specific styles directly on-device. On the other hand, while efficient finetuning techniques have many advantages, they bring the potential risk of digital forgery. This is mainly due to finetuning the generative models on a much smaller subset of data, leading to potential overfitting. As our proposed method is also a parameter-efficient finetuning technique, it suffers from similar potential risk as the other PEFT algorithms.

M   Limitations and Future Work

In this work, we show that our proposed sparse high rank adapter, SHiRA, with merely finetuning 1-2% parameters of the pretrained generative models is sufficient to achieve high performance on many adapter tasks. However, in order to adopt our method for mobile deployment, hardware-software

21

co-design techniques, such as lookup-table (LUT) based approaches, may be necessary to optimize the implementation for edge devices.

Moreover, as discussed in the main text, building optimal sparse masks (i.e., which parameters to train for a given task) warrants further investigation.
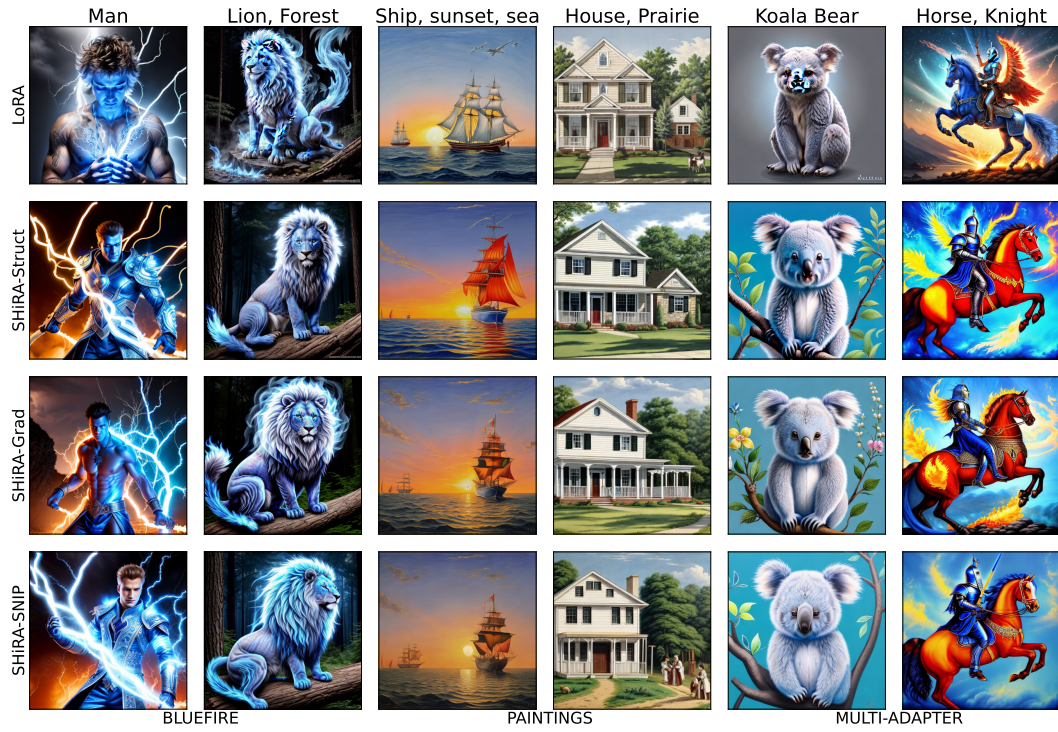
Figure 10: More image samples for single and multi-adapter fusion. We observe that LoRA exhibits artifacts for koala and concept loss for knight in Multi-Adapter fusion while SHiRA produces significantly better images.

22
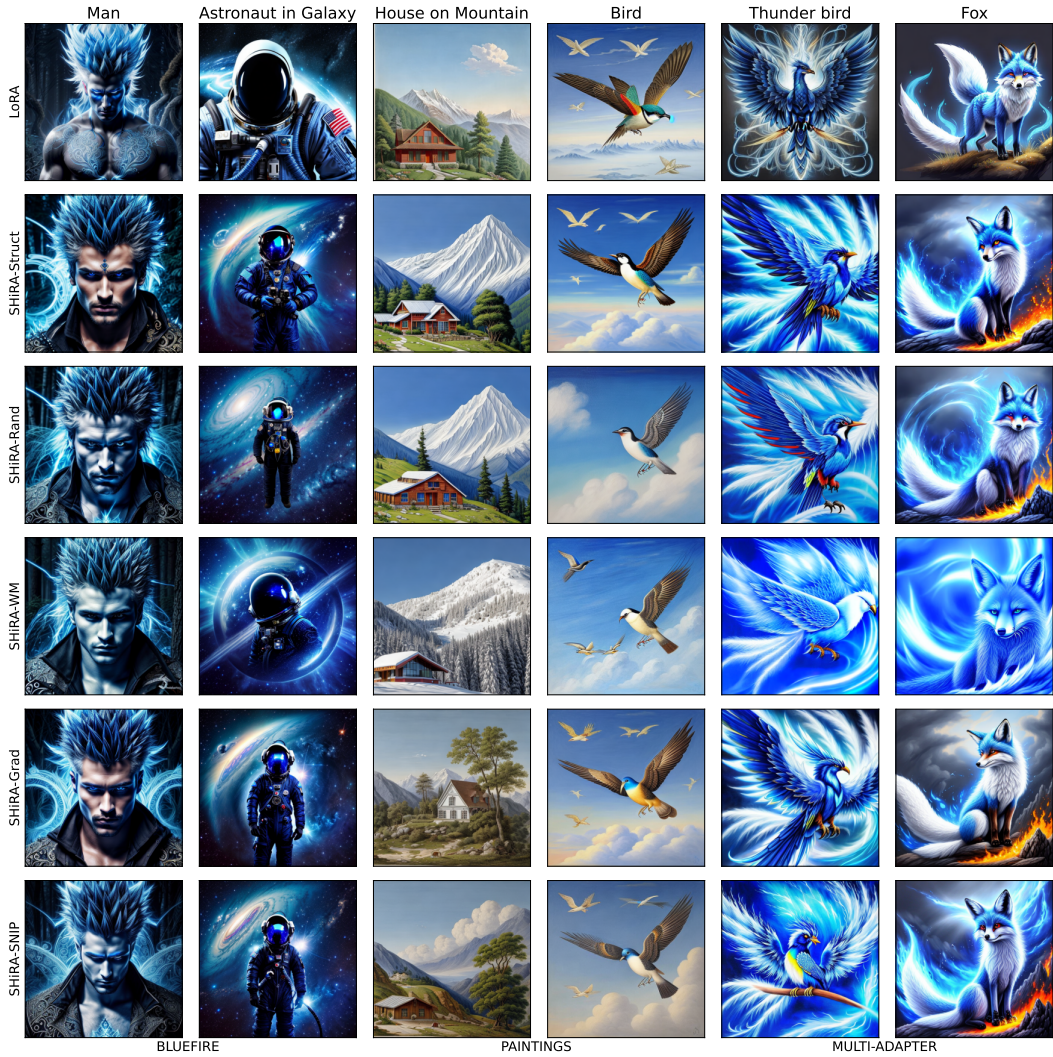
Figure 11: More image samples for single and multi-adapter fusion. We observe that LoRA images exhibit concept loss for bird in Multi-Adapter fusion.

23
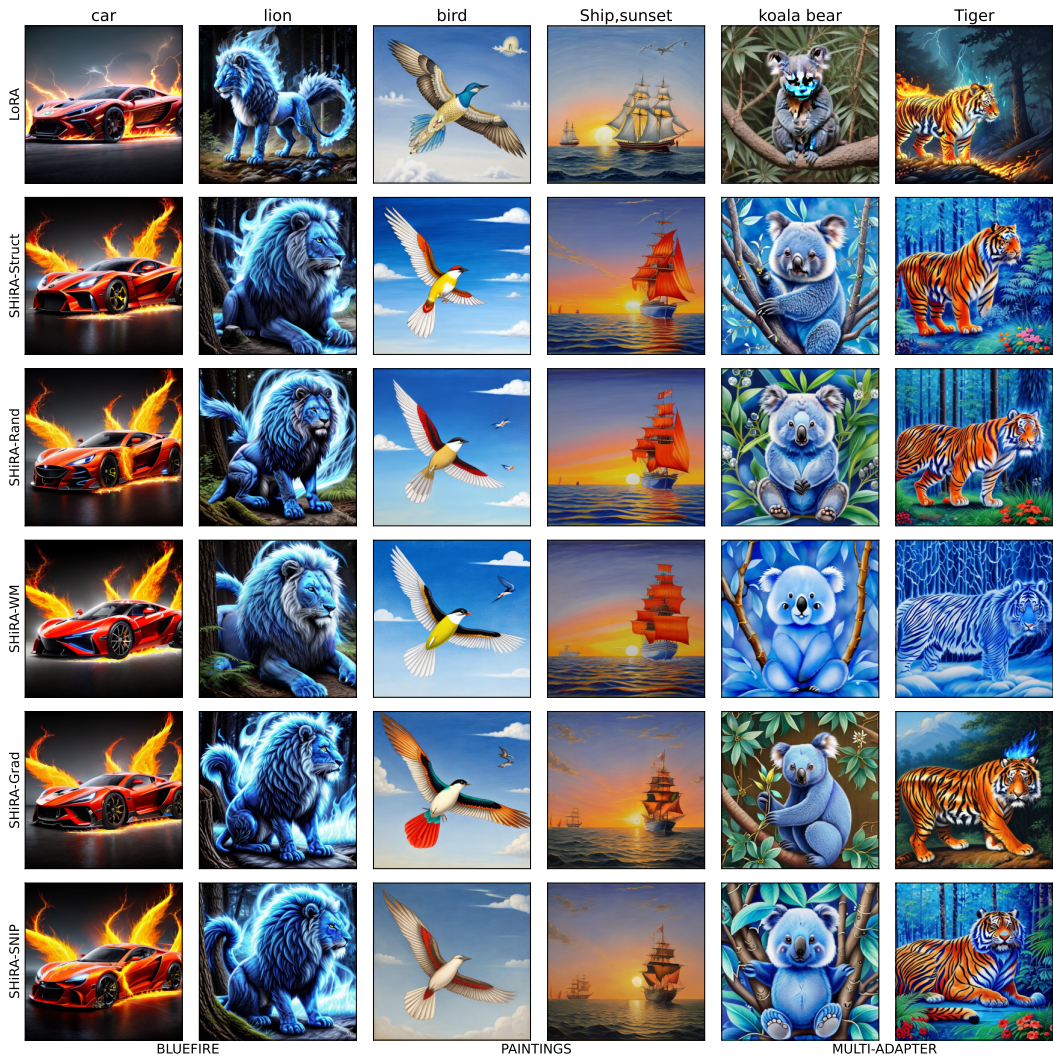
Figure 12: More image samples for single and multi-adapter fusion. Koala is not included in the training set of either of the Bluefire and Paintings Adapter styles. We observe that for this class, LoRA has significant artifacts whereas SHiRA produces exceptional images.

24
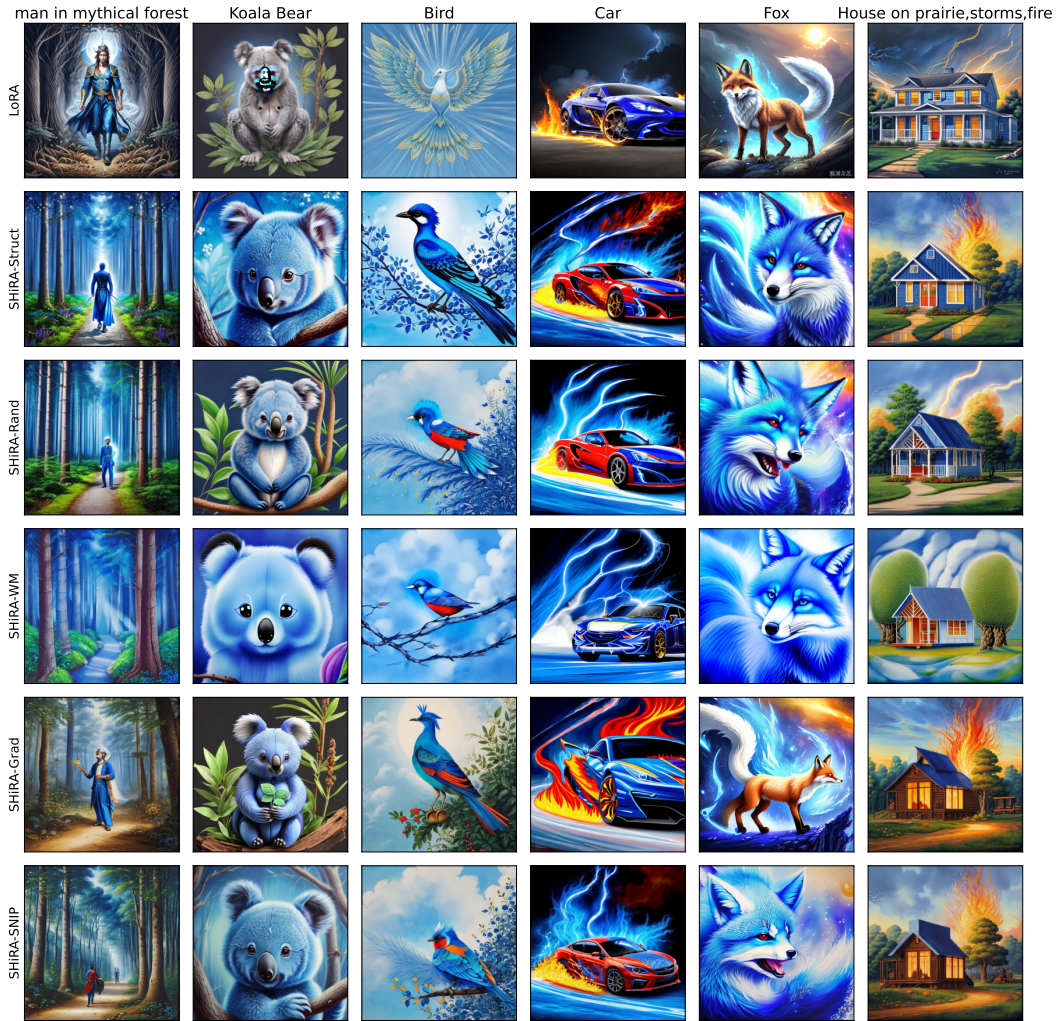
Figure 13: More results for multi-adapter fusion. Koala is not included in the training set of either of the Bluefire and Paintings Adapter styles. We observe that for this class, LoRA has significant artifacts whereas SHiRA produces exceptional images.

25

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The manuscript discusses and reports detailed results accurately reflecting the claims and the scope of the work.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Yes, discussed in section M.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

26

Justification: Discussed in section Appendix F.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: All experimentation details for training and inference are included in the main and supplementary materials.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

27

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: We plan to open source the code and datasets pending legal approval.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so âĂIJNoâĂİ is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: All experimentation details required for understanding the results are included in the main and supplementary materials.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Yes, mean and standard deviation of the performance metrics are reported across various seed values.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

28

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: Details of compute used for training and inference are included.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

Answer: [Yes]

Justification: We conform to NeurIPS code of ethics.

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: Yes, discussed in section Appendix L.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

29

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: Not applicable since our models do not have high risk of misuse.

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We follow the license terms for every model and dataset we use.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

30

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Yes, details of the datasets are provided in the Appendix G.1.2.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Not Applicable

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: Not Applicable

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

31