Rapid Switching and Multi-Adapter Fusion via Sparse High Rank Adapters

Kartikeya Bhardwaj [1 *] **Nilesh Prasad Pandey** [1 *] **Sweta Priyadarshi** [2] **Viswanath Ganapathy** [1] **Rafael Esteves** [1] **Shreya Kadambi** [1] **Shubhankar Borse** [1] **Paul Whatmough** [1] **Risheek Garrepalli** [1] **Mart Van Baalen** [1] **Harris Teague** [1] **Markus Nagel** [1]

Abstract

In this paper, we propose Sparse High Rank Adapters (SHiRA) that directly finetune 1-2% of the base model weights while leaving others unchanged, thus, resulting in a highly sparse adapter. This high sparsity incurs no inference overhead, enables rapid switching directly in the fused mode, and significantly reduces concept-loss during multi-adapter fusion. Our extensive experiments on LVMs and LLMs demonstrate that finetuning merely 1-2% parameters in the base model is sufficient for many adapter tasks and significantly outperforms Low Rank Adaptation (LoRA). We also show that SHiRA is orthogonal to advanced LoRA methods such as DoRA and can be easily combined with existing techniques.

1. Introduction

Low Rank Adaptation (LoRA) (Hu et al., 2021) 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. Despite its success, there are still several challenges from the standpoint of deployment on the edge. First, on mobile devices, 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) (Huggingface, 2024) inference latency while enabling *rapid adapter switching* in the unfused mode (see Appendix A). Second, as shown by many previous works (Yu et al., 2023; Shah et al., 2023; Gu et al., 2024), LoRA also exhibits *concept-loss* when multiple adapters are used concurrently. Finally, recent literature also contributes important theoretical and empirical knowledge towards the value of *high rank adapters*. For instance, Kalajdzievski (Kalajdzievski, 2023) shows that the high rank adapters can significantly outperform low rank adapters when used with correct scaling factors. This calls for further investigation into whether other high rank adapters would outperform LoRA.

Figure 1. Sparse High Rank Adapters (SHiRA): Changing ∼1-2% weights of the pretrained 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 the majority of parameters when fused, prohibiting rapid switching on mobile devices and also experiences concept loss/artifacts during multi-adapter fusion.

To address these challenges, in this paper, we propose Sparse High Rank Adapters (SHiRA), a single solution to all the problems discussed above. Specifically, we make the following **key contributions**: (*i*) We propose a new high rank adapter paradigm and demonstrate that changing as few as 1-2% parameters of the original network is sufficient for many adaptation tasks. (*ii*) We also conduct extensive experiments on LLaMA-7B, LLaMA2-7B, and Stable Diffusion models and demonstrate that SHiRA significantly outperforms LoRA on both single and multi-adapter fusion tasks (see Fig. 1). On LLMs, we show that SHiRA achieves up to 2.7% higher accuracy than LoRA on commonsense reasoning and is also complementary to more advanced variants of LoRA such as DoRA (Liu et al., 2024). (*iii*) We further provide a latency- and memory-efficient implementation based on Parameter-Efficient Finetuning (PEFT) Library for SHiRA which trains nearly as fast as standard LoRA while consuming lower peak GPU memory (see Appendix D). All our experiments can be run on a single NVIDIA-A100 GPU.

2. Related Work

LoRA, its variants, and sparse adapters. Many LoRA variants exist in literature: DoRA (Liu et al., 2024), LoRA+ (Hayou et al., 2024), VeRA (Kopiczko et al., 2023),

[*]Equal contribution [1]Qualcomm AI Research. Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc. [2]Work done while employed at Qualcomm AI Research. Correspondence to: Kartikeya Bhardwaj <kbhardwa@qti.qualcomm.com>, Nilesh Prasad Pandey <nileshpr@qti.qualcomm.com>.

Published at ICML 2024 Workshop on Foundation Models in the Wild.

LoRA-FA (Zhang et al., 2023), RS-LoRA (Kalajdzievski, 2023), among many others. The crucial difference between this literature and our work is that we develop a high rank sparse adapter which updates a fraction of weights in the pretrained weight tensor. A few other LoRA variants have also explored a combination of sparsity and low rank adaptation (Nikdan et al., 2024; Ding et al., 2023; He et al., 2022). Most of these proposed methods promote sparsity in the adapters through either pruning (He et al., 2022) or adaptive rank selection (Ding et al., 2023). However, since they combine their method with LoRA, the fused adapter weight still overwrites the entire pretrained weight tensor, hence prohibiting rapid adapter switching.

Partial Finetuning. Our work is most closely related to partial finetuning techniques proposed in the pre-LoRA era. These methods use either fixed sparse masks (Sung et al., 2021) or learned masks (Zhao et al., 2020; Guo et al., 2020) to finetune a pretrained network. Note that, partial finetuning techniques have been mostly explored for relatively *small* language models, and *not* for recent LLMs and diffusion models. 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 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.

Multi-Adapter Fusion. Existing methods like (Gu et al., 2024; Yu et al., 2023; Shah et al., 2023) use the base LoRA with non-trivial postprocessing (Yu et al., 2023; Shah et al., 2023). In contrast, we introduce a new adapter where multiple concepts naturally do not interfere with each other. Our work is also orthogonal to the prior multi-adapter fusion work and can be combined with these techniques.

3. Proposed Approach

3.1. Sparse High Rank Adapters (SHiRA)

SHiRA exploits highly sparse trainable parameters in the pretrained base model. 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 ($\sim$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). As a result, very few parameters get updated during training and our adapter consists of just those sparse weights. Various strategies to create the mask $\mathcal{M}$ are given below:

1. **SHiRA-Struct:** In this structured mask, only certain rows or columns of the weight as well as its diagonal are set to be trainable. The diagonal makes the mask high rank whereas the structured trainable rows/ columns lead to a rank 1 adapter, making the mask a combination of a sparse high rank and a rank 1 adapter.
2. **SHiRA-Rand:** This mask is obtained by randomly setting 1-2% parameters trainable.
3. **SHiRA-WM:** Top-K parameters are trained based on their absolute weight magnitudes (WM) for each layer.
4. **SHiRA-Grad:** Top 1-2% weights that receive the highest gradient magnitudes based on a calibration set are trained for each layer.
5. **SHiRA-SNIP:** This mask is based on SNIP metric used in the pruning literature (Lee et al., 2018). SNIP combines weight magnitude and gradient strategies, i.e., gradient magnitude times the weight magnitude.

3.2. Rapid Switching and Multi-Adapter Fusion

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 10$\times$ faster than equivalent LoRA fusing** (see Appendix B and Fig 5).

Next, we discuss multi-adapter fusion in SHiRA. Suppose, we are given two adapters $\mathcal{A}_1$ and $\mathcal{A}_2$ that represent two separate concepts. Now, consider the product $\mathcal{A}_1^T \mathcal{A}_2$ to evaluate the relative orthogonality of two adapters. Note that, for SHiRA, the adapters are based on sparse masks $\mathcal{M}_1$ and $\mathcal{M}_2$ which are both up to 98-99% sparse. Due to this high sparsity of SHiRA masks, $\mathcal{A}_1^T \mathcal{A}_2$ contains a much higher number of zeros compared to equivalent dense LoRA adapters (since fused LoRA adapter AB is a dense matrix). Therefore, we hypothesize that this high sparsity property of SHiRA adapters would result in significantly lower interference between adapters and, hence, reduced concept loss. We empirically validate this in section 4.

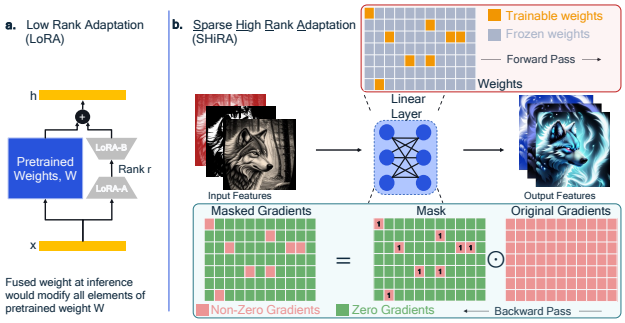

Figure 2. (a) LoRA appends two low rank weights that can be fused into the base weights at inference. However, this modifies all weights and prevents rapid switching. (b) SHiRA finetunes very few pretrained weights by exploiting gradient-masking during training. We show that finetuning as low as 1-2% parameters is sufficient to achieve high accuracy on many adapter tasks.

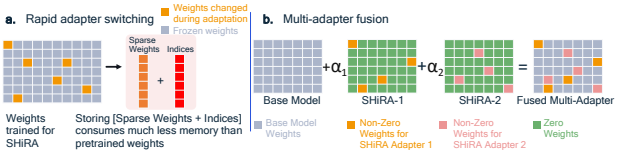

Figure 3. (a) Rapid switching: SHiRA adapters can be stored as sparse weights and their indices which can be loaded on the base model. Since only 1-2% weights need to be overwritten, the adapter can be efficiently switched with different weights at inference, eliminating the need for a separate fusion stage. (b) Multi-adapter fusion: Multiple adapters can be fused together by naively adding them together and then loading the resulting sparse weights.

Style	Method	%Params	HPSv2 score(↑) $\alpha = 1$	HPSv2 score(↑) $\alpha = 0.5$
Paintings	LoRA	3.84	24.7 ± 1.8	31.3 ± 1.5
	SHiRA-Struct	**1.99**	$\mathbf{31.2 \pm 1.7}$	$\mathbf{33.0 \pm 1.8}$
	SHiRA-Rand	2.05	30.7 ± 1.9	32.7 ± 1.9
	SHiRA-WM	2.05	29.7 ± 1.9	32.1 ± 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**	$\mathbf{34.2 \pm 1.6}$	$\mathbf{34.1 \pm 1.5}$
	SHiRA-Rand	2.05	33.4 ± 1.9	33.7 ± 1.7
	SHiRA-WM	2.05	31.9 ± 2.1	33.1 ± 1.7
	SHiRA-Grad	2.05	$\mathbf{34.2 \pm 1.5}$	33.7 ± 1.7
	SHiRA-SNIP	2.05	33.7 ± 1.7	33.7 ± 1.6

Table 1. Comparison between LoRA and SHiRA schemes with respect to HPSv2 metric.

3.3. Memory- and Latency-Efficient SHiRA Training.

We can have two implementations: (*i*) backward hook based gradient masking to turn any trainer into SHiRA training (refer Appendix C) (*ii*) PEFT based implementation. As discussed in Appendix D, 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. Experiments

4.1. 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 E for more details). For both these datasets, we conduct single adapter and multi-adapter experiments and quantify the image quality using Human Preference Score-V2 (HPSv2) (Wu et al., 2023).

On the language domain, we experiment with LLaMA 7B (Touvron et al., 2023a), LLaMA2-7B (Touvron et al., 2023b) and evaluate it on various commonsense reasoning tasks such as HellaSwag, PIQA, SIQA, BoolQ, Arc-easy, Arc-challenge, OpenBookQA and Winogrande. Specifically, we follow the setup adopted by (Hu et al., 2023; Liu et al., 2024) for training and evaluating LoRA (Hu et al., 2021), DoRA (Liu et al., 2024), and SHiRA on downstream tasks. Also, similar to our vision investigations, we conduct single and multi-adapter experiments on LLMs as well.

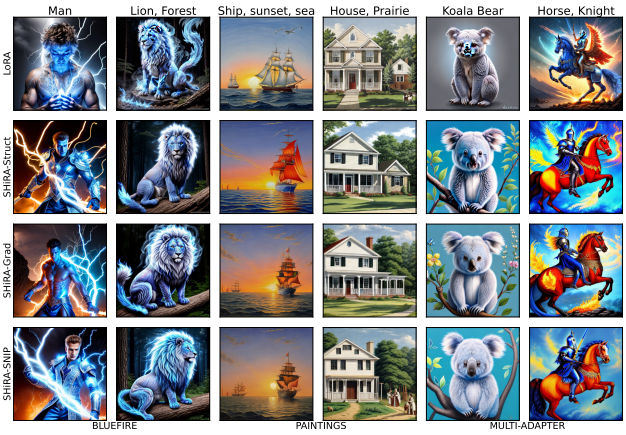

Figure 4. Comparison between different SHiRA masking methods for single and multi adapter image generation. For multi-adapter fusion, SHiRA-Struct outperforms all other adapters. SHiRA does not have artifacts and concept-loss like LoRA (see Koala/Knight).

4.2. Vision Results

4.2.1. 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 1-2% pretrained weights, SHiRA generates high quality images for both finetuning tasks.

Next, we compare various types of SHiRA masks in Fig. 4. 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

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 models 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 over baselines. *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.

	Single Adapter				Multi-Adapter				
Model	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	78.07	**79.71**	**77.57**	**78.45**	**77.43**	**76.88**	**67.76**	**74.02** (+6.69%)	4.43

Table 4. Multi-adapter fusion of independently trained SHiRA and LoRA adapters on BoolQ, PIQA, and Arc-Easy for LLaMA2-7B. %Drop is calculated as drop in average accuracy for multi-adapter fusion compared to the single adapter average accuracy for each adapter.

or similar HPSv2 scores than LoRA, especially for larger α, the scaling factor used for modifying the intensity of the adapter during inference (see Appendix G).

4.2.2. SHiRA Improves Multi-Adapter Fusion

We validate the effectiveness of various SHiRA schemes for multi-adapter fusion. In Fig. 1 and 4, both SHiRA-Struct and SNIP (right two columns) are clearly better at capturing both concepts than LoRA. For example, the knight image in Fig. 4 generated with LoRA seems to lose most of the paintings concept (more results in Fig. 7, Appendix H).

4.3. Language Results

4.3.1. Single Adapter SHiRA Finetuning

For LLMs, different SHiRA adapters are trained on the combined 170K sample commonsense reasoning dataset released by (Hu et al., 2023; Liu et al., 2024). Similar to (Liu et al., 2024), we train our SHiRA adapters for 3 epochs and compare it against the LoRA baselines. As shown in Table 2 for LLaMA-7B, 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 adapter switching on edge devices. Of note, since SHiRA-Struct is a combination of rank-1 adapter and a sparse diagonal adapter, it did not perform well on more complex LLM tasks.

SHiRA is orthogonal to newer adapters like DoRA and can be easily integrated with them. To demonstrate this, we create a high rank weight decomposed adapter similar to DoRA and then mask 99% of this weight using SHiRA. As shown 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. Finally, we experiment with LLaMA2-7B and demonstrate that SHiRA-SNIP – which achieved the best results on LLaMA-7B – yields significant gains compared to LoRA and nearly the same accuracy as DoRA (within 0.4%, see Table 3).

4.3.2. Multi-Adapter Fusion on LLMs

We now extend our analysis 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 4.3.1 was trained on a combined dataset containing 170K samples from all eight commonsense benchmarks, as proposed in (Hu et al., 2023; Liu et al., 2024). In the present section, the goal is to evaluate how much accuracy drop various adapters experience after multi-adapter fusion. Due to its simplicity towards constructing a mask, we use SHiRA-WM which consists of top 1% parameters being trained for all tasks. As shown in table 4, multi-SHiRA-WM outperforms multi-LoRA on all the three benchmarks, outperforming LoRA by 6.69% accuracy on average.

5. 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 demonstrate SHiRA's ability to rapidly switch adapters and to avoid concept loss with naive multi-adapter fusion, contrary to LoRA which suffers from both of these problems. We have further conducted extensive single- and multi-adapter experiments on several vision and language tasks and demonstrate the effectiveness of SHiRA compared to 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 up to $10\times$ faster than equivalent LoRA fusion on a CPU, thus demonstrating our rapid switching benefits.

For further reading, please refer to our extended work (Bhardwaj et al., 2024).

References

Bhardwaj, K., Pandey, N. P., Priyadarshi, S., Ganapathy, V., Esteves, R., Kadambi, S., Borse, S., Whatmough, P., Garrepalli, R., Van Baalen, M., et al. Sparse high rank adapters. *arXiv preprint arXiv:2406.13175*, 2024.

Ding, N., Lv, X., Wang, Q., Chen, Y., Zhou, B., Liu, Z., and Sun, M. Sparse low-rank adaptation of pre-trained language models. *arXiv preprint arXiv:2311.11696*, 2023.

Gu, Y., Wang, X., Wu, J. Z., Shi, Y., Chen, Y., Fan, Z., Xiao, W., Zhao, R., Chang, S., Wu, W., et al. Mix-of-show: Decentralized low-rank adaptation for multi-concept customization of diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024.

Guo, D., Rush, A. M., and Kim, Y. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463*, 2020.

Hayou, S., Ghosh, N., and Yu, B. Lora+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*, 2024.

He, S., Ding, L., Dong, D., Zhang, M., and Tao, D. Sparseadapter: An easy approach for improving the parameter-efficiency of adapters. *arXiv preprint arXiv:2210.04284*, 2022.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Hu, Z., Wang, L., Lan, Y., Xu, W., Lim, E.-P., Bing, L., Xu, X., Poria, S., and Lee, R. K.-W. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*, 2023.

Hudson, D. A. and Manning, C. D. Gqa: A new dataset for real-world visual reasoning and compositional question answering, 2019.

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

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

Kopiczko, D. J., Blankevoort, T., and Asano, Y. M. Vera: Vector-based random matrix adaptation. *arXiv preprint arXiv:2310.11454*, 2023.

Lee, N., Ajanthan, T., and Torr, P. H. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

Liu, S.-Y., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., and Chen, M.-H. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*, 2024.

Nikdan, M., Tabesh, S., and Alistarh, D. Rosa: Accurate parameter-efficient fine-tuning via robust adaptation. *arXiv preprint arXiv:2401.04679*, 2024.

Podell, D., English, Z., Lacey, K., Blattmann, A., Dockhorn, T., Müller, J., Penna, J., and Rombach, R. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

Shah, V., Ruiz, N., Cole, F., Lu, E., Lazebnik, S., Li, Y., and Jampani, V. Ziplora: Any subject in any style by effectively merging loras. *arXiv preprint arXiv:2311.13600*, 2023.

Sung, Y.-L., Nair, V., and Raffel, C. A. Training neural networks with fixed sparse masks. *Advances in Neural Information Processing Systems*, 34:24193–24205, 2021.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Wu, X., Hao, Y., Sun, K., Chen, Y., Zhu, F., Zhao, R., and Li, H. Human preference score v2: A solid benchmark for evaluating human preferences of text-to-image synthesis. *arXiv preprint arXiv:2306.09341*, 2023.

Yu, L., Yu, B., Yu, H., Huang, F., and Li, Y. Language models are super mario: Absorbing abilities from homologous models as a free lunch. *arXiv preprint arXiv:2311.03099*, 2023.

Zhang, L., Zhang, L., Shi, S., Chu, X., and Li, B. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*, 2023.

Zhao, M., Lin, T., Mi, F., Jaggi, M., and Schütze, H. Masking as an efficient alternative to finetuning for pretrained language models. *arXiv preprint arXiv:2004.12406*, 2020.

A. 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 adapter switching since it will increase DRAM traffic considerably; (*ii*) keep the adapter unfused and run the adapter in unfused mode: this can help with rapid adapter switching but would incur significant additional latency (up to 30% higher) as shown in (Huggingface, 2024), since we would have LoRA branches in the forward pass during inference; (*iii*) use the Huggingface/Diffusers pipeline (Huggingface, 2024) which is used for server-grade GPUs. This pipeline consists of `load`→`fuse`→`inference`→`unfuse`→`unload` to switch adapters. In the last option, 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.

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 (Huggingface, 2024) and iteratively add adapters to SDXL model (Podell et al., 2023). 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, none of the deployment options presented above are feasible for rapid adapter 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` (Huggingface, 2024) 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. 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. 5, our `scatter_op`-based SHiRA inference pipeline is up to $10\times$ faster than fusing LoRA weights, specially for larger dimensions.

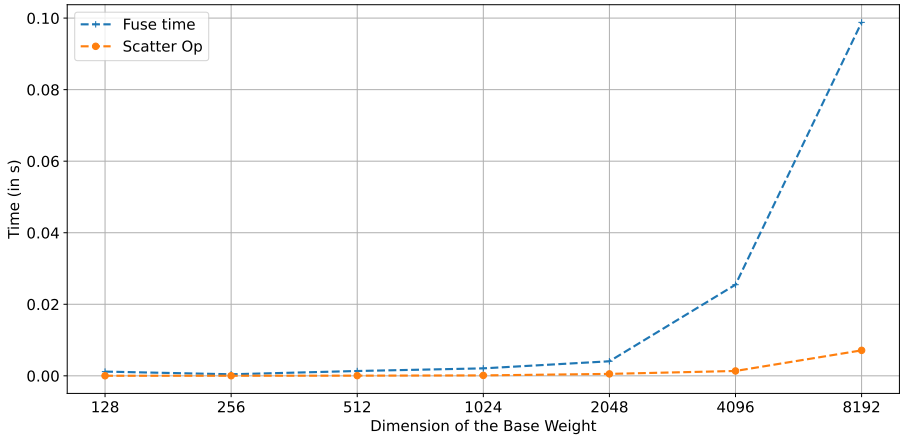

Figure 5. Comparison between average times for LoRA-fuse and SHiRA-`scatter_op` implementation for 10 randomly initialized weights of various dimensions on a CPU (e.g., dimension = 4096 means that the weight has shape 4096×4096). For fusing, we compute time taken to merge LoRA adapters into the base 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.

C. Turn any Trainer into SHiRA: Gradient Hook based Implementation

In this section, we provide a method to adapt 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.

D. Latency- and Memory-Efficient PEFT based Implementation for SHiRA

As discussed in Appendix B, `scatter_op`-based implementation 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 6, 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 lower ($\sim 40\%$) peak GPU memory and also trains much faster (SHiRA is about 36% faster than DoRA). Finally, note that, partial finetuning techniques proposed in the pre-LoRA era do not have such memory-efficient implementations, which made them impractical for large generative models. All memory requirement data was collected using `psutil` utility used within the `Transformers.Trainer` training loop.

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 6. Peak GPU memory consumption (in GBs) and Training steps per second during training for various PEFT-based implementation of 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).

E. Dataset and Evaluation Metric Descriptions

E.1. Datasets

E.1.1. Language Datasets

For language finetuning tasks, we use the commonsense reasoning datasets, which comprise 8 sub-tasks, each with a predefined training and testing set as shown in table 7. We follow the setting of (Hu et al., 2023) for SHiRA Single Adapter training. The common sense reasoning training dataset is a combination of the training datasets provided by (Hudson & Manning, 2019), 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 7. Commonsense Benchmark

E.1.2. Vision Datasets

For style transfer adaptation tasks as described in sections 4.2.1 and 4.2.2, 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*.

E.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.

F. Training Details

In this section, we list hyperparameters used for our experiments for Language and Vision finetuning tasks in table 8.

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

Table 8. Training hyperparameters used for finetuning experiments.

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

G. Effect of Alpha

Figure 6. Effect of different α on SHiRA based Bluefire image generation.

As described in the 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 amplified 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. 6 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.

H. More Results

We show many more sample images for multi-adapter fusion in Fig. 7.

Figure 7. 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.