Low-Rank Interconnected Adaptation across Layers

Yibo Zhong[1], Jinman Zhao[2], Yao Zhou[1*†],

[1]Sichuan University, [2]University of Toronto,
zhongyibo@stu.scu.edu.cn yaozhou@scu.edu.cn

Abstract

Low-rank adaptation (LoRA) is a widely used parameter-efficient fine-tuning (PEFT) method that learns weight updates $\Delta W = AB$ for pre-trained weights W through low-rank adapters A and B. While LoRA ensures hardware efficiency, its low-rank weight updates limit adaptation performance. In this paper, we propose low-rank interconnected adaptation across layers (Lily), a novel PEFT method that introduces an interconnected framework with locally shared A and globally shared B experts. This structure eliminates redundant per-layer AB pairs, enabling higher-rank ΔW with equal or fewer parameters. To enhance expressiveness, we use data-dependent routers to determine A-B interconnections, preventing B experts from converging to the same behavior and improving representational power across domains. Experiments across modalities, architectures, and model sizes demonstrate Lily's superior performance and efficiency. Github

1 Introduction

Fine-tuning foundation models like Transformers (Vaswani et al., 2017) on downstream tasks is common but costly, especially for large models like LLMs, which incur high computational and storage demands and risk catastrophic forgetting (Biderman et al., 2024). Linear probing alleviates these issues by fine-tuning only the final modules, but suffers from performance loss due to frozen backbone weights. To address this, parameter-efficient fine-tuning (PEFT) freezes the backbone and introduces lightweight modules for task-specific learning. Among PEFT methods, Low-rank Adaptation (LoRA (Hu et al., 2021)) is widely used, particularly for LLMs. LoRA introduces low-rank projection matrices, A and B, to approximate weight updates ΔW, achieving significant savings in computation and storage while outperforming linear probing by updating the backbone weights.

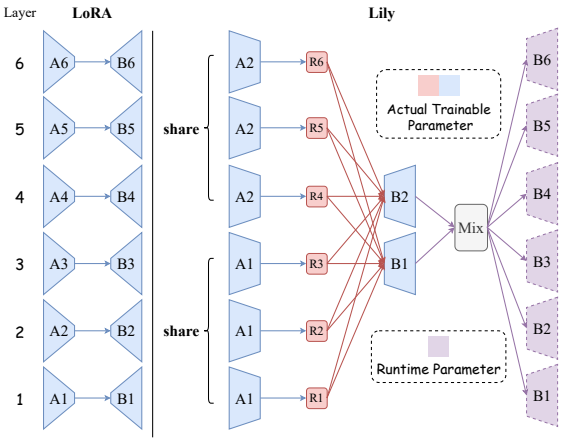

Figure 1: Dynamics of LoRA and Lily. In this 6-layer example with a fixed overall parameter budget, LoRA allocates the same parameter budget to each layer, resulting in small rank updates for the weights. Lily overcomes this by employing a small number of shared adapters with a much larger rank, achieving higher-rank updates while using the same or even a smaller parameter budget. Considering the different characteristics, and to make the adaptation more dynamic, the adapters are mixed according to a data-dependent router, represented by R.

However, LoRA and its subsequent improvements (Miles et al., 2024; Zhang et al., 2023; Zhong et al., 2024) face a limitation: the learned weight updates ΔW are constrained to be low-rank, limiting model performance. A key issue is that LoRA allocates the same parameter budget to each layer, regardless of their importance (Fig. 1). As a result, the rank of each adapter is constrained by the fixed budget, raising the critical question: *Can we enable more dynamic, expressive adaptation with high-rank weight updates under the same parameter budget?*

In this paper, we propose **L**ow-rank **i**nterconnected adaptation across **l**a**y**ers (Lily), a novel framework for more expressive and

*Corresponding author

†This work was supported by National Natural Science Foundation of China (Grant 62376172)

Findings of the Association for Computational Linguistics: ACL 2025, pages 17005–17029
July 27 - August 1, 2025

efficient PEFT. Specifically, we decouple A and its corresponding upward B, eliminating their tight coupling. Each A is connected to all Bs, and vice versa, as illustrated in Fig. 1. This creates a hierarchical structure where locally-shared As perform downward projections at specific layers, while globally-shared Bs perform upward projections across all layers. To enhance dynamism, we selectively connect each A with Bs based on layer features. A extracts features from the current layer, and a selective mixture of Bs is performed based on these features, enabled by routers (Shazeer et al., 2017) that generate data-dependent weight distributions for B experts.

The interconnected structure makes the adaptation process more dynamic and flexible, with rich interactions between adapters. By reducing the number of adapters and increasing their rank, **Lily achieves higher-rank weight updates than LoRA while using the same or fewer parameters.** Additionally, Lily enables comprehensive information access and learning by allowing adapters at each layer to collaborate, share knowledge, and model dependencies across layers. Our key contributions include:

- We propose Lily, a novel PEFT framework that introduces interconnected adapters, effectively overcoming the limitations of low-rank weight updates in LoRA under the same parameter constraints.
- Lily utilizes routers to dynamically select and connect an adapter A with multiple adapter B experts, enabling richer information flow and more expressive adaptation dynamics.
- Extensive experiments are conducted across diverse modalities, architectures, and model scales, demonstrating Lily's superior performance and efficiency in a wide range of scenarios.

2 Related Work

Parameter Efficient Fine-Tuning Foundation models are typically pre-trained on large datasets and fine-tuned on downstream tasks. Parameter-efficient fine-tuning (PEFT) seeks to fine-tune models efficiently with minimal parameters while maintaining performance and preserving learned knowledge. It effectively addresses limitations of conventional fine-tuning techniques, like full fine-tuning or linear probing. Current PEFT approaches can be divided into two categories: 1) adapter-based methods (Hu et al., 2021; Chen et al., 2022; Pfeiffer et al., 2020a; Jie and Deng, 2023; Houlsby et al., 2019a; Zhong and Zhou, 2025) and 2) prompt-based methods (Tu et al., 2023a,b). Adapter-based methods insert lightweight adapters into the Multi-Head Self-Attention (MHSA) or Feed-Forward Network (FFN) blocks of the Transformer architecture, while prompt-based methods add trainable tokens to the input sequence.

Among these, low-rank adaptation (LoRA (Hu et al., 2021)) is a well-known technique. It introduces projection matrices A and B for each adaptation target W, where A projects input x to a low-dimensional space and B restores it to the original dimension. The product of these matrices approximates the weight update ΔW in full fine-tuning (FFT). However, this limits the update to a low-rank subspace, which may affect performance. Additionally, A and B are tightly coupled, restricting the adaptation process to information from the current layer, which may hinder the modeling of dependencies across layers.

Mixture of Experts Mixture of Experts (MoE) is an active research area that has received significant attention, especially in the field of large language models (LLMs). Conditional computation, which activates different parts of the network on a per-example basis, has been proposed to enhance model capability without increasing computation (Davis and Arel, 2013; Bengio et al., 2013; Eigen et al., 2013; Almahairi et al., 2016) . The sparsely-gated MoE layer is introduced to implement this idea, consisting of numerous sub-networks (Shazeer et al., 2017). A trainable gating network (router) determines the combination of experts for each example. There are already PEFT methods like MoLORA (Zadouri et al., 2023) and MOLA (Gao et al., 2024a) that apply the MoE design concept to PEFT. However, these methods simply treat the adapters A and B combined in LoRA as a single expert. Concurrent research Wu et al. (2024) utilizes A and B sub-spaces as the experts but fails to overcome the limitation discussed in the previous section. Another concurrent work, HydraLoRA Tian et al. (2024), explores an asymmetric design for LoRA. Unlike our work, we consider the interconnection across layers and deploy a model-wide asymmetric design to enable cross-layer connections. This enables the use of adapters of higher rank than a typical LoRA setup while using the same or fewer overall parameters.

3 Methodology

3.1 Downward Projection and Selective Weight Allocation

The process is illustrated in the right half of Fig. 1. Initially, we use an A to project the input $x \in \mathbb{R}^{N \times C_{\text{in}}}$ into its low-dimensional representation $x' \in \mathbb{R}^{N \times d}$, where N is the sequence length:

$$x' = xA \tag{1}$$

To enable more parameter efficiency, the number of As can be set to less than the number of layers in the model by sharing the same A across neighboring layers, as illustrated in Fig. 1 and discussed in A. Inspired by the Mixture of Experts (MoE) paradigm, we employ a router $R \in \mathbb{R}^{N_e \times d}$ to selectively assign weights to all B experts based on their relationship to the current layer's features (x'), where N_e represents the number of B experts. A weight set $S \in \mathbb{R}^{N_e}$ is obtained as:

$$S = \text{softmax}\left(\sum_{i=1}^{N}(x'R^T)_i\right) \tag{2}$$

The router selectively mixes the experts based on this data-dependent weight distribution, enabling information integration and expressive adaptation.

3.2 Weighted Mixture of Experts and Upward Projection

Once we obtain the low-dimensional input x', we combine information from all layers using the model-wide shared B experts. One intuitive approach is to feed x' into each B expert and combine their outputs to obtain the additional knowledge $x_\Delta \in \mathbb{R}^{N \times C_{\text{out}}}$. However, to address efficiency concerns discussed in Appendix A.2, we propose an alternative implementation that is mathematically equivalent but significantly reduces the computational burden, described as follows:

$$x_\Delta = x'\left(\sum_{i=1}^{N_e} S_i \cdot B^i\right) \tag{3}$$

where S is the set of weight scores for the B experts, obtained through selective weight allocation. Since each S_i is a scalar value, the calculation in Eq. 3 is mathematically equivalent to the intuitive method but with significantly improved efficiency. Therefore, the complete computation flow, with input $x \in \mathbb{R}^{N \times C_{\text{in}}}$ and output $y \in \mathbb{R}^{N \times C_{\text{out}}}$, for an adaptation target module is:

$$y = xW_0 + s \cdot x_\Delta \tag{4}$$

where s is a scaling factor. By selectively allocating weights and mixing B experts, Lily enables access to all levels of information during adaptation. Each layer's target adaptation modules can consider the status and knowledge from all other layers, resulting in a more expressive and comprehensive adaptation. Meanwhile, thanks to its interconnectivity, Lily can break the low-rank update constraint of LoRA by simply employing a smaller number of adapters with higher ranks.

4 Experiments

We validate the effectiveness of Lily across different domains, model sizes (from ViT to LLM), and architectures (Transformers, Mamba), demonstrating its generally strong adaptation capability. Concurrently, we conduct a comprehensive analysis of Lily's intrinsic mechanisms, providing a thorough understanding of how it works. All ranks for Lily are selected from 8, 16, 32, ensuring that the total parameter count does not exceed that of the baselines. All experiments are conducted on a single RTX 4090 GPU. Additionally, multiple analyses are provided in Appendix C, D, E, F, G, H, I, and J.

4.1 Common Sense Reasoning

Implementation: We evaluate Lily on commonsense reasoning with LLMs. For the implementation, we utilize LLaMA3-8B (AI@Meta, 2024) and Falcon-Mamba-7B (Zuo et al., 2024) as backbones. LLaMA3 is a near-SOTA open-source large language model, while Falcon-Mamba is an open-sourced large language model based on the Mamba architecture. Using these models allows us to validate the effectiveness of Lily for fine-tuning LLMs and assess whether this effectiveness can be transferred to architectures beyond Transformers (Mamba, in this case). We fine-tune these models on Commonsense170K (Hu et al., 2023) and evaluate the adaptation results on eight multiple-choice problem tasks, including BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-e, ARC-c (Clark et al., 2018), and OBQA (Mihaylov et al., 2018). The compared methods are LoRA for Falcon-Mamba and LoRA (Hu et al.,

Table 1: Commonsense reasoning results for Falcon-Mamba-7B across eight tasks. Bold represents the highest performance for each dataset utilizing PEFT methods. "Δ" and "in" refer to adaptations of Mamba's `delta_proj` and `in_proj` parameters, respectively.

Model	PEFT	Params	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
ChatGPT	-	-	73.1	85.4	68.5	78.5	66.1	89.8	79.9	74.8	77.0
Falcon-Mamba-7B	LoRA	3.7M	6.5	30.5	40.6	**14.9**	56.4	42.2	31.8	38.4	32.7
	Lily (Δ + in)	3.7M	44.9	**66.8**	65.0	10.5	57.1	78.7	64.6	**68.2**	57.0
	Lily (in)	**3.3M**	**60.2**	61.0	**67.3**	12.9	**61.5**	**80.0**	67.5	65.8	**59.5**

Table 2: Commonsense reasoning results for LLaMA3-8B across eight tasks. † represents results taken from Liu et al. (2024) and (Wang et al., 2024). Bold denotes the highest performance scores for each dataset among different PEFT methods.

Model	PEFT	Params	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
ChatGPT	-	-	73.1	85.4	68.5	78.5	66.1	89.8	79.9	74.8	77.0
LLaMA3-8B	LoRA†	56M	70.8	85.2	**79.9**	91.7	84.3	84.2	71.2	79.0	80.8
	PiSSA†	83.8M	67.1	81.1	77.2	83.6	78.9	77.7	63.2	74.6	75.4
	MiLoRA†	56.6M	68.8	**86.7**	77.2	**92.9**	**85.6**	86.8	75.5	81.8	81.9
	Lily	**1.2M**	**72.9**	85.6	77.8	92.7	83.3	**89.7**	**77.6**	**82.8**	**82.8**

2021), PiSSA (Meng et al., 2024), and MiLoRA (Wang et al., 2024) for LLaMA3. We only compare LoRA for Falcon-Mamba because tailored PEFT methods for Mamba-based LLMs have not yet been proposed, which is beyond the scope of this paper. Detailed hyper-parameter settings and dataset information are reported in Appendix B.1.1 and Appendix B.2.1.

Results We report the accuracy in Tables 2 and 1. Based on these results, it can be observed that Lily outperforms the other compared PEFT methods with a smaller parameters budget. Specifically, Lily surpasses LoRA by a significant margin on Falcon-Mamba and, on LLaMA3, outperforms both LoRA and MiLoRA. This demonstrates Lily's superior adaptation capability and parameter efficiency in handling commonsense reasoning tasks. Additionally, although performance on Falcon-Mamba is notably lower than that of the baseline and LLaMA3, we believe this discrepancy stems from the inherent limitations of the model rather than any deficiency in Lily, as Lily still significantly outperforms LoRA on Falcon-Mamba while demonstrating robust performance on LLaMA3. These findings also highlight that current state of Mamba-based LLMs generally exhibits inferior performance compared to Transformer-based LLMs such as ChatGPT and LLaMA on many tasks.

4.2 Natural Language Understanding

Implementation We evaluate Lily on natural language understanding (NLU) tasks. For the implementation, we use RoBERTa Base (Liu et al., 2019) and RoBERTa Large as the backbones and fine-tune them on tasks from the GLUE benchmark (General Language Understanding Evaluation (Wang et al., 2018)), which consists of multiple NLU tasks, including single-sentence classification, similarity and paraphrase, and natural language inference tasks. We compare Lily against several competitive PEFT methods, including BitFit (Zaken et al., 2021), Adapter-Tuning (Rücklé et al., 2020; Houlsby et al., 2019b; Lin et al., 2020; Pfeiffer et al., 2020b), LoRA (Hu et al., 2021), DyLoRA (Valipour et al., 2022), FLoRA (Hao et al., 2024), and AdaLoRA (Zhang et al., 2023). Additionally, we utilize full fine-tuning (FFT) as the baseline. Specific hyperparameters and dataset information are provided in Appendix B.1.2 and B.2.2.

Results The results are shown in Table 3. From the table, we can clearly observe that Lily surpasses all the compared PEFT methods by a significant margin, demonstrating its ability to tackle NLU tasks. Among the six tasks, Lily surpasses FFT on four of them when using RoBERTa Base and RoBERTa Large, showcasing its strong approximation ability and high parameter efficiency.

4.3 Subject-driven Image Generation

Implementation We conduct experiments on fine-tuning text-to-image diffusion models for the subject-driven generation task. As the backbone, we use SDXL and fine-tune it using both LoRA and Lily. First, we fine-tune the model on images paired with text prompts (e.g., "A photo of a [v] duck toy"), each of which includes a unique identifier. Afterward, text prompts containing the identifier are used to generate customized images.

Table 3: Various fine-tuning methods applied to RoBERTa Base and RoBERTa Large are evaluated on six datasets from the GLUE benchmark. We present the Matthews correlation coefficient (MCC) for CoLA, the Pearson correlation coefficient (PCC) for STS-B, and accuracy (Acc.) for the remaining tasks. The highest performance for each dataset is highlighted in **bold**, with all metrics favoring higher values across the six datasets.

Model & Method	# Trainable Parameters	**SST-2** (Acc.)	**MRPC** (Acc.)	**CoLA** (MCC)	**QNLI** (Acc.)	**RTE** (Acc.)	**STS-B** (PCC)	**Avg.**
RoB_{base}(FFT)	125M	94.8	90.2	63.6	92.8	78.7	91.2	85.2
RoB_{base}(BitFit)	0.1M	93.7	**92.7**	62	91.8	81.5	90.8	85.4
RoB_{base}(Adpt^{D})	0.3M	94.2	88.5	60.8	93.1	71.5	89.7	83.0
RoB_{base}(Adpt^{D})	0.9M	94.7	88.4	62.6	93.0	75.9	90.3	84.2
RoB_{base}(LoRA)	0.3M	94.8	89.8	63.3	92.9	78.2	**91.5**	85.1
RoB_{base}(AdaLoRA)	0.3M	94.5	88.7	62.0	**93.1**	81.0	90.5	85.0
RoB_{base}(DyLoRA)	0.3M	94.3	89.5	61.1	92.2	78.7	91.1	84.5
RoB_{base}(FLoRA)	0.3M	91.2	85.8	65.4	92.3	65.0	87.6	81.2
RoB_{base}(**Lily**)	0.3M	**95.0**	90.2	**66.0**	92.5	**81.6**	90.8	**86.0**
$\text{RoB}_{\text{large}}$(FF)	356M	**96.4**	**90.9**	68	94.7	86.6	**92.4**	88.2
$\text{RoB}_{\text{large}}$($\text{Adpt}^{\text{H}}$)	0.8M	96.3	87.7	66.3	94.7	72.9	91.5	84.9
$\text{RoB}_{\text{large}}$(LoRA)	0.8M	96.2	90.2	68.2	**94.8**	85.2	92.3	87.8
$\text{RoB}_{\text{large}}$(Lily)	0.5M	95.6	**90.9**	**68.4**	**94.8**	**88.4**	91.9	**88.4**

Results The results are presented in Fig. 2 following the format in Gao et al. (2024b) and Wu et al. (2024). From these results, we observe that the images generated by Lily generally align better with the text prompts. For instance, when asked to generate an image of a duck toy floating on water, Lily's output accurately depicts the designated environment, whereas LoRA's does not. Additionally, when asked to generate an image of a wolf plushie in the snow, Lily precisely captures the snow around the wolf, while LoRA fails to do so. These observations demonstrate Lily's excellent performance in text-to-image generation with more expressive adaptation. Additional generated results are provided in Appendix I.

4.4 Visual Adaptation Benchmark

Implementation We assess Lily on the Visual Task Adaptation Benchmark (VTAB-1K (Zhai et al., 2019)), a suite of 19 visual tasks spanning diverse domains and semantics, to test its general visual adaptation capability. Tasks are categorized into Natural, Specialized, and Structured, and are all formulated as classification problems for consistent model evaluation. We conduct two sets of experiments: one focusing on adaptation effectiveness on the Vision Transformer (ViT (Dosovitskiy et al., 2020)) and the other on Vision Mamba (Vim (Zhu et al., 2024)), demonstrating Lily's architecture-agnostic capabilities. For ViT, we use ViT-B pre-trained on ImageNet-21K (Deng et al., 2009), and for Vim, we use Vim-s pre-trained on ImageNet-1K. To fairly compare ViT and Vim architectures, we implement LoRA (Hu et al., 2021) and AdaptFormer (Chen et al., 2022) on ViT-B pre-trained on ImageNet-1K. In the ViT experiments, we compare Lily with LoRA, AdaptFormer, FourierFT (Gao et al., 2024b), and MoRA (Jiang et al., 2024); in the Vim experiments, we focus on contrasting architectural differences and, therefore, use only LoRA as the baseline. All experiments include full fine-tuning (FFT) and linear probing as baselines. For Vim, we implement two versions of Lily: Lily-S (Small) and Lily-L (Large), with different hyper-parameter settings to either reduce the parameter count (Lily-S) or maximize performance (Lily-L). For Lily on ViT, the reported results are obtained from adapting both the self-attention and the MA module in the Transformer. Regarding the performance of the fine-tuned module, we conduct additional experiments in Appendix D. Detailed experimental settings and dataset information are provided in Appendix B.1.3 and B.2.3.

Results The results are shown in Tables 4 and 5. For ViT, Lily significantly outperforms all compared PEFT methods while also offering improved parameter efficiency. In contrast, the performance on the Vim backbone is generally lower than that on ViT; for instance, LoRA on ViT performs better than LoRA on Vim. We argue that this difference is due to variations in architecture design and overall model size. However, Lily's strong adaptation performance allows it to match or exceed the performance of other PEFT methods on ViT and to significantly outperform LoRA on Vim

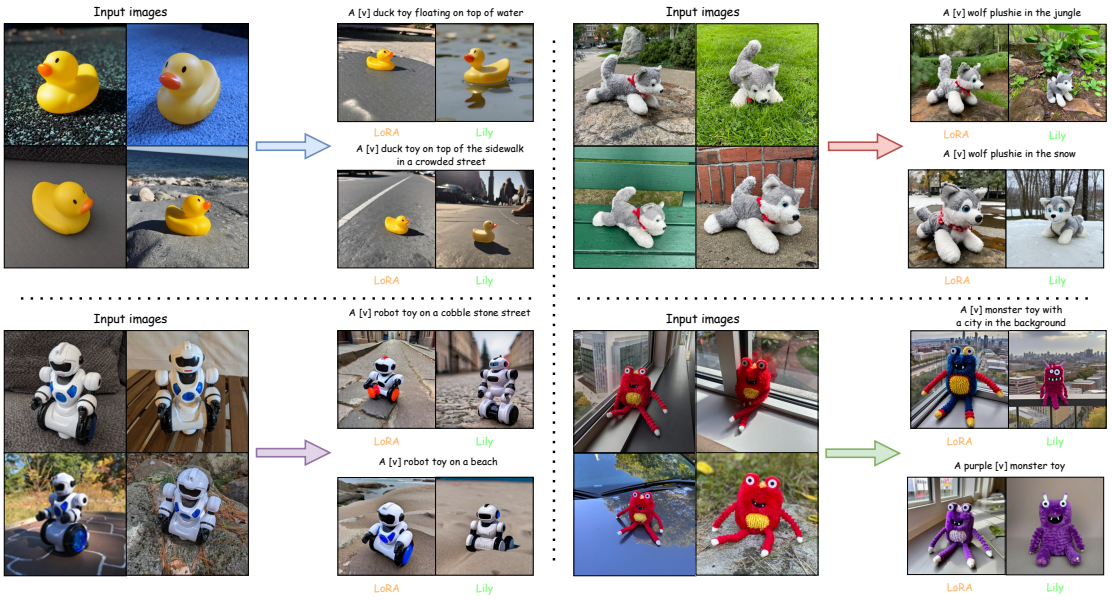

Figure 2: Qualitative results of subject-driven generation. Lily's results align better with prompts, featuring more accurate color, environment, and shape.

(with both Lily-S and Lily-L surpassing LoRA by a significant margin). This demonstrates Lily's architecture-agnostic capability, highlighting its potential across various model architectures. Overall, Lily achieves excellent visual adaptation performance while maintaining architecture-agnosticity and high parameter efficiency.

4.5 Understanding Lily

4.5.1 Does It Have High-Rank Weight Updates?

The interconnected and asymmetric structure of Lily enables a flexible allocation of the parameter budget, thereby allowing weight updates with higher ranks across all layers. To validate this claim, we provide an empirical analysis, as shown in Fig. 3. Specifically, we run four tasks from the NLU experiment and measure the rank of the weight updates for the query transformation matrix W_q in the first three layers. We use a small number of matrices A and B (2 or 3) with a rank of 32 to match the parameter count of LoRA, which uses adapters with a rank set to 8. Specific hyperparameter settings can be found in Appendix B.1.2.

From the results, we observe that the rank of the weight updates from Lily is generally notably larger than that of LoRA when using a similar number of parameters. Meanwhile, the weight updates from Lily still exhibit a higher rank compared to those of LoRA even when using only 16.7% of

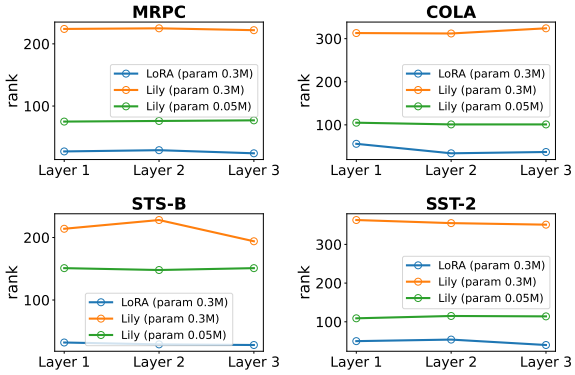

Figure 3: Actual rank of the weight updates. The weight updates are of shape 768×768. We run 20 epochs for COLA, MRPC, and STS-B, and 3 epochs for SST-2. It can be easily observed that the weight updates from Lily have notably higher rank than those from LoRA. Note that the reported rank is computed from accumulated weight updates over multiple epochs.

LoRA's parameters. This empirical analysis essentially validates our claim that Lily achieves high-rank updates with the same parameter budget. We attribute this to the model-wide sharing and the cross-layer asymmetric design, which facilitate a flexible allocation of the parameter budget.

4.5.2 What's the Influence of Adapter Granularity?

The number of experts in the model-wide B module can be freely set, and the number of As can also be flexibly determined by sharing across the

Table 4: Full results of Lily on ViT-B pre-trained on ImageNet-21K for the VTAB-1K benchmark, with averages computed based on group-wise results. **Bold** indicates the best performance.

			Natural							**Specialized**				**Structured**							
	Params(M)	Average	**Cifar100**	**Caltech101**	**DTD**	**Flowers102**	**Pets**	**SVHN**	**Sun397**	**Camelyon**	**EuroSAT**	**Resisc45**	**Retinopathy**	**Clevr-Count**	**Clevr-Dist**	**DMLab**	**KITTI-Dist**	**dSpr-Loc**	**dSpr-Ori**	**sNORB-Azim**	**sNORB-Ele**
Conventional Fine-Tuning																					
FFT	86	68.9	68.9	87.7	64.3	97.2	86.9	87.4	38.8	79.7	95.7	84.2	73.9	56.3	58.6	41.7	65.5	57.5	46.7	25.7	29.1
LP	0	57.6	64.4	85.0	63.2	97.0	86.3	36.6	51.0	78.5	87.5	68.5	74.0	34.3	30.6	33.2	55.4	12.5	20.0	9.6	19.2
PEFT methods																					
AdaptFormer	0.588	76.8	74.0	92.2	71.7	**99.3**	**91.7**	88.9	56.4	87.2	95.1	**85.7**	**75.9**	**84.2**	62.2	53.0	81.0	87.1	53.6	35.3	42.3
Bi-LoRA	1.180	76.7	72.1	91.7	71.2	99.1	91.4	**90.2**	55.8	87.0	**95.4**	85.5	75.5	83.1	64.1	52.2	81.3	86.4	53.5	36.7	44.4
LoRA	1.180	76.4	72.5	91.5	71.9	99.1	91.4	89.6	56.0	87.6	95.3	84.0	75.0	83.6	64.3	51.6	80.9	86.0	51.8	**36.8**	42.3
FourierFT	0.936	72.7	69.1	88.8	71.9	99.0	91.0	79.0	55.6	84.9	93.0	83.2	74.9	70.7	61.1	45.2	74.8	78.0	53.0	24.8	30.8
MoRA	1.058	75.4	72.1	90.0	71.7	99.2	91.1	90.1	56.0	87.1	94.8	85.1	75.4	76.7	62.3	49.7	78.3	83.1	53.0	34.5	34.5
Lily	0.318	**77.3**	**73.9**	**93.0**	**72.9**	**99.3**	91.6	89.0	**56.6**	**87.9**	95.2	84.9	75.7	83.9	**65.4**	**53.4**	**81.6**	**88.2**	**54.5**	**37.0**	**45.4**

Table 5: Full results of Lily on Vim-S pre-trained on ImageNet-1K for the VTAB-1K benchmark, with averages calculated within each group. * denotes linear probing results from Tu et al. (2023a). For fair comparison, we also use ViT-B pre-trained on ImageNet-1K. **Bold** indicates best performance among Vim-based PEFT methods.

			Natural							**Specialized**				**Structured**							
	Params(M)	Average	**Cifar100**	**Caltech101**	**DTD**	**Flowers102**	**Pets**	**SVHN**	**Sun397**	**Camelyon**	**EuroSAT**	**Resisc45**	**Retinopathy**	**Clevr-Count**	**Clevr-Dist**	**DMLab**	**KITTI-Dist**	**dSpr-Loc**	**dSpr-Ori**	**sNORB-Azim**	**sNORB-Ele**
Conventional Fine-Tuning																					
FFT-Vim	26	70.1	47.7	89.4	64.2	89.0	87.7	90.6	35.1	84.5	93.9	81.0	74.5	67.5	52.9	47.3	78.9	75.3	53.9	33.3	29.4
FFT-ViT	86	69.9	49.4	89.3	65.5	91.7	89.1	91.4	33.5	85.9	93.6	85.4	74.3	54.7	55.2	48.7	79.7	68.2	49.7	31.5	27.7
LP-Vim	0	55.3	40.9	83.3	57.3	66.3	86.3	38.4	34.6	79.0	87.6	65.0	73.6	36.3	35.1	33.3	64.8	23.0	21.6	15.1	21.7
LP-ViT	0	66.4	50.6	85.6	61.4	79.5	86.5	40.8	38.0	79.7	91.5	71.7	65.5	41.4	34.4	34.1	55.4	18.1	26.4	16.5	24.8
PEFT on ViT																					
AdaptFormer	0.147	72.4	56.2	89.6	67.2	91.2	91.1	85.9	42.1	85.4	94.6	84.0	74.3	75.8	58.6	48.6	79.6	81.6	53.7	29.6	35.2
LoRA	0.295	72.5	56.4	89.0	66.9	91.2	90.4	86.9	41.5	85.4	95.1	84.1	75.2	75.8	61.7	47.7	80.5	80.4	52.0	29.4	35.7
PEFT on Vim																					
LoRA	0.054	70.1	57.5	87.7	64.4	86.0	90.0	85.7	39.8	82.2	93.8	79.6	72.5	78.6	56.5	42.0	80.5	71.8	51.0	28.4	32.6
Lily-S	0.074	71.4	**58.2**	88.5	65.6	87.1	**90.7**	87.5	40.4	83.3	94.1	79.7	73.8	81.2	57.3	44.1	80.9	79.3	54.1	30.0	33.7
Lily-L	0.196	**72.3**	57.8	**89.4**	**66.2**	**87.8**	90.5	**88.1**	**40.5**	**84.1**	**94.3**	**81.3**	**75.1**	**81.6**	**57.8**	**46.5**	**81.0**	**82.9**	**55.2**	**32.1**	**34.8**

same level of layers, as introduced in Appendix A.1. Therefore, we analyze the impact of these choices on performance. We denote the number of A experts and B experts as *ne*_1 and *ne*_2, respectively. For simplicity, we set them equal in the experiments and denote this common value as ***ne***. We refer to the number of layers each expert attends to as adapter granularity. As the value of *ne* increases, the adapter granularity becomes finer. As shown in Fig. 5, the results from the VTAB-1K benchmark indicate different patterns. For instance, on the DTD dataset, the best performance is achieved when *ne* is 4, while on sNORB-Azim, performance increases as *ne* increases. Increasing *ne* leads to more parameters and finer adapter granularity; however, finer adapter granularity does not necessarily translate to better overall performance. For example, on Resisc45, DTD, Cifar100, sNORB-Ele, dsPr-LoC, Flowers102, and EuroSAT, the negative impact of increasingly finer adapter granularity eventually outweighs the benefits of the additional parameters, leading to a decrease in overall performance. In other tasks, different patterns may occur because the positive effect of adapter granularity on performance is consistently strong, or its negative effect is insufficient to offset the benefits of increased parameters, resulting in generally improved performance with higher *ne*. This phenomenon provides an important insight: for most tasks, simply increasing the number of parameters may not lead to better performance. Instead, only when adapter granularity and the number of param-

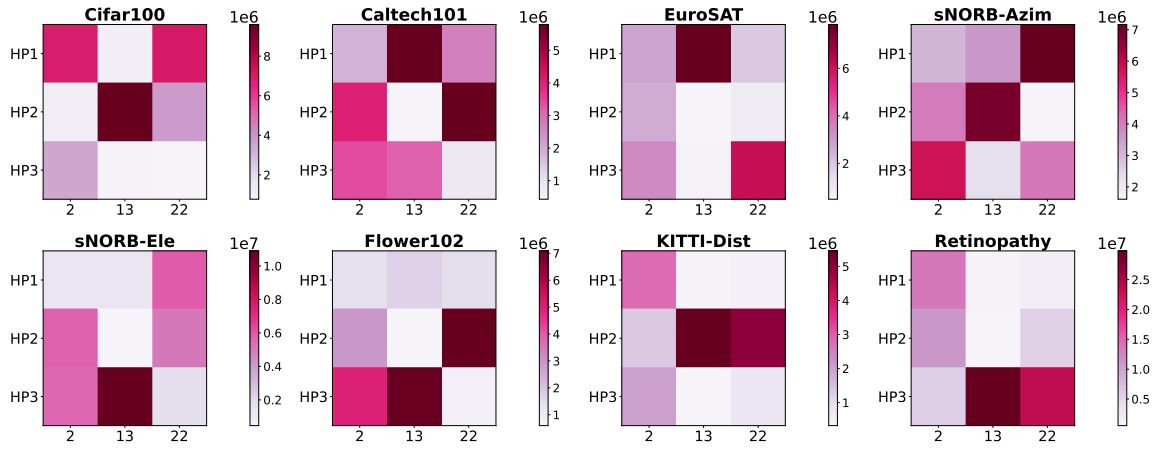

Figure 4: Visualization of accumulated assigned weight for B experts by a router across various layers. Example here uses layer of index 2, 13 and 22 to represent shallow, middle and deep layers. The reported values are based on the accumulated router outputs over multiple epochs.

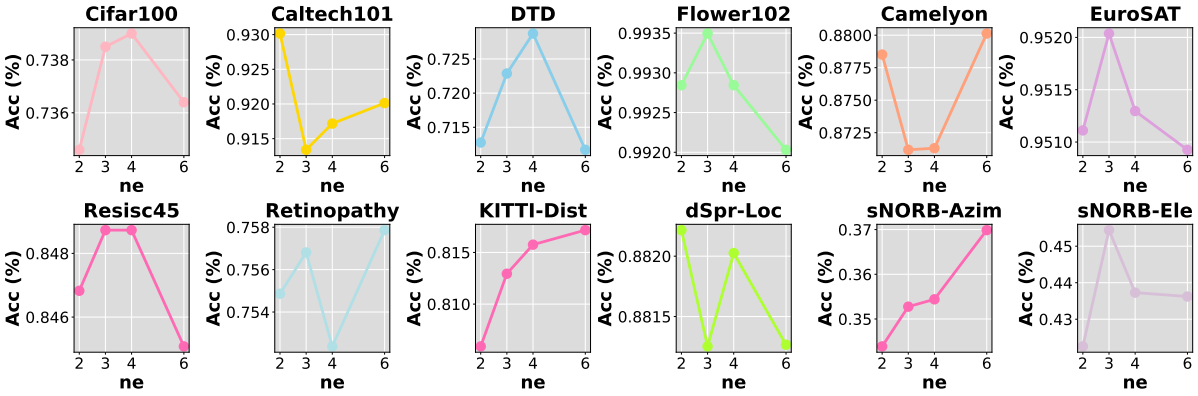

Figure 5: Impact of attention granularity (i.e., the choice of how many As and Bs) on the performance. We choose 12 out of 19 tasks from VTAB-1K for a comprehensive understanding.

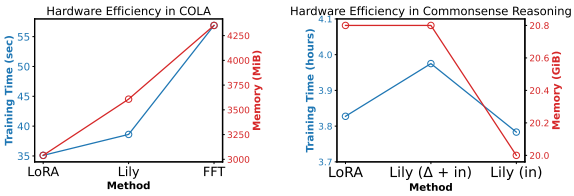

Figure 6: Hardware efficiency of Lily compared to LoRA. We run 10 epochs for COLA. We report the training time and memory consumption. It can be observed that Lily generally performs on par with LoRA in terms of hardware efficiency.

eters reach an optimal trade-off can we achieve the best performance.

4.5.3 Does It Exhibit Selectivity?

Lily uses routers to assign varying weights to different B experts, thereby achieving selective information combination. We illustrate this selectivity in Fig. 4. We use a setup with three B experts and select three layer levels (1, 13, 22) to calculate the total weight assigned to each expert. The results reveal clear selectivity: for different layers, the router assigns significantly different weights to the B experts. For instance, on Cifar100, the middle layer is predominantly dominated by B 2, whereas the deep layer is primarily dominated by B 1 and B 2. In contrast, on Retinopathy, both the middle and deep layers are dominated by B 3. This selectivity ensures that, even when different layers share information, the inherent differences between layers are still taken into account, making the adaptation more flexible and comprehensive.

4.5.4 What's the Hardware Efficiency?

The dynamics of Lily obviously introduce complexity to the design of LoRA. In this section, we analyze how this affects the hardware efficiency of Lily compared to LoRA. We use the COLA task from the NLU experiments with RoBERTa-Base and run for 10 epochs. Additionally, we also report

the runtime and GPU memory consumption in the Falcon-Mamba experiment.

The results are shown in Fig. 6, from which we can observe that the hardware efficiency of Lily is comparable to LoRA. Specifically, Lily slightly underperforms LoRA in the NLU experiment but performs on par with LoRA in the LLM experiment. In general, the introduced complexity of Lily does not prevent it from being an more effective PEFT method that is also hardware-friendly.

5 Conclusion

In this paper, we propose Low-Rank Interconnected Adaptation (Lily), a novel framework for efficient fine-tuning via the interconnectivity of adapters. Lily enables each layer to access information from others during adaptation through a hierarchical structure. Additionally, it successfully overcomes the low-rank update limitation of LoRA, enabling high-rank updates and, therefore, better adaptation capability under the same parameter budget. Our approach consistently improves performance across various modalities, model sizes, and architectures, surpassing existing methods while maintaining enhanced efficiency. In summary, Lily's versatility and efficiency make it a promising approach for a wide range of applications.

6 Limitations

Although Lily has been experimentally evaluated in a wide range of scenarios, we have not explored all possible applications where PEFT could be used. These potential areas are left as directions for future work.

7 Ethics Statement

This work is an improvement upon LoRA. However, it could potentially be used for fine-tuning diffusion models or large language models (LLMs) for generating malicious content.

References

AI@Meta. 2024. Llama 3 model card.

Amjad Almahairi, Nicolas Ballas, Tim Cooijmans, Yin Zheng, Hugo Larochelle, and Aaron Courville. 2016. Dynamic capacity networks. In *International Conference on Machine Learning*, pages 2549–2558. PMLR.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Dan Biderman, Jose Gonzalez Ortiz, Jacob Portes, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, et al. 2024. Lora learns less and forgets less. *arXiv preprint arXiv:2405.09673*.

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

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

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

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

Andrew Davis and Itamar Arel. 2013. Low-rank approximations for conditional feedforward computation in deep neural networks. *arXiv preprint arXiv:1312.4461*.

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

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. 2013. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314*.

Chongyang Gao, Kezhen Chen, Jinmeng Rao, Baochen Sun, Ruibo Liu, Daiyi Peng, Yawen Zhang, Xiaoyuan Guo, Jie Yang, and VS Subrahmanian. 2024a. Higher layers need more lora experts. *arXiv preprint arXiv:2402.08562*.

Ziqi Gao, Qichao Wang, Aochuan Chen, Zijing Liu, Bingzhe Wu, Liang Chen, and Jia Li. 2024b. Parameter-efficient fine-tuning with discrete fourier transform. *arXiv preprint arXiv:2405.03003*.

Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Yongchang Hao, Yanshuai Cao, and Lili Mou. 2024. Flora: Low-rank adapters are secretly gradient compressors. *arXiv preprint arXiv:2402.03293*.

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

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

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

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

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

Shibo Jie and Zhi-Hong Deng. 2023. Fact: Factor-tuning for lightweight adaptation on vision transformer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 1060–1068.

Shibo Jie, Haoqing Wang, and Zhi-Hong Deng. 2023. Revisiting the parameter efficiency of adapters from the perspective of precision redundancy. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17217–17226.

Zhaojiang Lin, Andrea Madotto, and Pascale Fung. 2020. Exploring versatile generative language model via parameter-efficient transfer learning. *arXiv preprint arXiv:2004.03829*.

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

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. Peft: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft.

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

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*.

Roy Miles, Pradyumna Reddy, Ismail Elezi, and Jiankang Deng. 2024. Velora: Memory efficient training using rank-1 sub-token projections. *arXiv preprint arXiv:2405.17991*.

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

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

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2020. Adapterdrop: On the efficiency of adapters in transformers. *arXiv preprint arXiv:2010.11918*.

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

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

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Chunlin Tian, Zhan Shi, Zhijiang Guo, Li Li, and Chengzhong Xu. 2024. Hydralora: An asymmetric lora architecture for efficient fine-tuning. *arXiv preprint arXiv:2404.19245*.

Cheng-Hao Tu, Zheda Mai, and Wei-Lun Chao. 2023a. Visual query tuning: Towards effective usage of intermediate representations for parameter and memory efficient transfer learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7725–7735.

Cheng-Hao Tu, Zheda Mai, and Wei-Lun Chao. 2023b. Visual query tuning: Towards effective usage of intermediate representations for parameter and memory efficient transfer learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7725–7735.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

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

Taiqiang Wu, Jiahao Wang, Zhe Zhao, and Ngai Wong. 2024. Mixture-of-subspaces in low-rank adaptation.

Ted Zadouri, Ahmet Üstün, Arash Ahmadian, Beyza Ermiş, Acyr Locatelli, and Sara Hooker. 2023. Pushing mixture of experts to the limit: Extremely parameter efficient moe for instruction tuning. *arXiv preprint arXiv:2309.05444*.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

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

Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, et al. 2019. The visual task adaptation benchmark.

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

Yibo Zhong and Yao Zhou. 2025. Rethinking low-rank adaptation in vision: Exploring head-level responsiveness across diverse tasks. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 7787–7796. IEEE.

Zihan Zhong, Zhiqiang Tang, Tong He, Haoyang Fang, and Chun Yuan. 2024. Convolution meets lora: Parameter efficient finetuning for segment anything model. *arXiv preprint arXiv:2401.17868*.

Lianghui Zhu, Bencheng Liao, Qian Zhang, Xinlong Wang, Wenyu Liu, and Xinggang Wang. 2024. Vision mamba: Efficient visual representation learning with bidirectional state space model. *arXiv preprint arXiv:2401.09417*.

Jingwei Zuo, Maksim Velikanov, Dhia Eddine Rhaiem, Ilyas Chahed, Younes Belkada, Guillaume Kunsch, and Hakim Hacid. 2024. Falcon mamba: The first competitive attention-free 7b language model.

Appendix

A More discussion about Lily

A.1 Model Structure and Design Intuition of Lily

Within the overall framework of Lily, we delve into specific implementation details and model design insights. First, we establish the relationship between A and B: A is confined to specific levels of layers, capturing features that enable the router to selectively assign weights to the B experts. In contrast, B is a model-wide module comprising multiple experts, each of which contains information from a particular level of layers.

We highlight several key aspects that are not heavily discussed in the methodology section:

A.1.1 Number of As

Since A is limited to specific layers, the simplest approach would be to place an A at each layer of the module to be adapted (e.g., the query transformation in MHSA). However, this setup may not be necessary, as the importance of each layer varies, and many layers have significantly lower importance than others (Zhang et al., 2023).

To achieve greater parameter efficiency, we can use fewer As, with each A focusing on a level of layers rather than a single layer. For example, an A can focus on shallow layers (e.g., layers 0, 1, 2, etc.) or deep layers. To enable a single A to handle multiple layers, we can share an A across multiple layers. By doing so, we eliminate the redundancy of placing an A at each layer, reduce the number of parameters, and improve efficiency.

This is exactly the strategy adopted in most of the experiments.

A.1.2 Number of B Experts

Regarding B, the number of experts can be set arbitrarily, allowing for more flexible configurations. In our experiments, for the sake of simplicity, we set the number of B experts equal to the number of As, thereby equating the granularity of A and B.

A.1.3 Routers Setup

There are multiple possible configurations for the router. First, we can bind the router to B, resulting in only one router per model. However, since the number of parameters in the router is relatively small, having only one router per model may not provide significant selectivity. Therefore, we can also bind the router to A, configuring a separate router for each A.

Most of our experiments use the latter setup. However, in the vision experiments on Vim, we adopt the single-router and no-lp-sharing setup to evaluate its effectiveness. The results indicate that this setup also performs well.

As future work, we can further verify the effectiveness of the latter setup on Vim, which may potentially lead to superior performance.

A.1.4 Hyperparameters

We detail the hyperparameters used in Lily. Specifically, we use **Lily_r** to represent the hidden dimension of the projectors: As and Bs. It serves the same function as r in LoRA. We use **Lily_s** to represent the scaling factor used by Lily, which is primarily searched within the range {0.01, 0.1, 1.0, 10.0, 100.0}.

We use ***ne*_1** to denote the number of As used in the model. Since As can be shared, as discussed in the previous section, *ne*_1 does not necessarily equal the number of layers in the model. Similarly, we use ***ne*_2** to represent the number of B experts in the model-wide B module.

In our experiments, we set *ne*_1 = *ne*_2 to enhance parameter efficiency and maintain simplicity.

A.1.5 Design Intuition

Lily employs a hierarchical structure to enable updates with higher ranks than LoRA. However, simply connecting all Bs equally to As does not yield the best performance. From the perspective of feature and information utilization across layers, merely aggregating all Bs for an A ignores the distinctiveness of features from the current layers. Meanwhile, this approach reduces the variability in the combinations of gradient projection matrices (since S_i and $C_{i,j}$ become constants), making the rank of the weight update higher than that of LoRA (as multiple distinct random matrices are used), but still not high enough for optimal performance due to the lack of variability in the combination process.

To address this, we introduce selectivity into the interconnectivity, as discussed below, making the combination of Bs data-dependent. This ensures that each S_i is unique across time steps, enabling updates with even higher ranks. This approach is similar to that of Hao et al. (2024), where a random matrix is constantly resampled to maintain high-rank updates. We further analyze this in Appendix

G.

A.2 Efficient Implementation for Weighted Combination

A straightforward implementation of the weighted combination in Lily is to pass the inputs through all the experts and then sum the results. This approach requires N_e matrix multiplications, N_e scalar multiplications, and N_e matrix additions. Despite its intuitive nature, the computational burden of this method is quite substantial.

However, Eq. 3, which is adopted in Lily, requires only N_e scalar multiplications, N_e matrix additions, and a single matrix multiplication. This optimization eliminates approximately N_e matrix multiplications, which can significantly reduce computational costs as the model size and the number of adaptation targets increase.

For an input x' of size $\mathbb{R}^{N\times d}$ and a projection matrix $P_H \in \mathbb{R}^{d\times C}$, the floating-point operations (FLOPs) of these two implementations are:

$$\begin{aligned}\text{FLOPs} &= \sum_{i=1}^{N_e}(2NdC) + \sum_{i=1}^{N_e}(dC) + \sum_{i=1}^{N_e}(NC)\\ &= N_e \times (2NdC + dC + NC),\\ \text{FLOPs} &= 2\sum_{i=1}^{N_e}(dC) + 2NdC\\ &= 2dC \times (N + N_e),\end{aligned} \tag{5}$$

From this, we can easily observe that the approach adopted by Lily requires fewer computations, thereby improving both speed and efficiency during the fine-tuning process. Under the setting of $N = 1024, d = 16, C = 768, N_e = 4$, the FLOPs for the intuitive approach amount to 0.104 GFLOPs, whereas for Lily, it is merely 0.025 GFLOPs, potentially leading to a $4\times$ speedup.

A.3 Actual Implementation of Lily

We present the actual implementation of Lily in Fig. 7. In this example, we showcase its implementation for visual adaptation tasks, specifically in the VTAB-1K benchmark. For large language models (LLMs), the implementation is slightly more complex due to modifications to the Hugging Face PEFT library (Mangrulkar et al., 2022), but the fundamental adaptation process remains the same.

Specifically, given an input, we first use the corresponding A of the current layer to project it into a low-dimensional representation. This low-dimensional representation is then used to selectively assign weights to the B experts. Once the weights for all experts are obtained, we proceed to combine these B experts accordingly, as discussed in Appendix A.2. After the weighted combination, the combined B is used to project the low-dimensional representation back into a high-dimensional space, thereby incorporating the additional knowledge gained through adaptation.

B Experimental Settings

B.1 Hyperparameters

A detailed description of the hyperparameters used in Lily is provided in Appendix A.1.

B.1.1 Commonsense Reasoning

The hyperparameters used in commonsense reasoning experiments for MiLoRA and PiSSA are provided in Tables 7 and 6. The settings for Lily and LoRA using Falcon-Mamba as the backbone are presented in Tables 9 and 8.

Notably, Lily achieves the best performance by adapting only the multi-head self-attention (MHSA) module in LLaMA3-8B, whereas other compared methods adapt all modules, including MA. Moreover, Lily utilizes the fewest parameters, demonstrating its superior adaptation capability in low-parameter-budget scenarios.

B.1.2 Natural Language Understanding

The specific hyperparameter settings for Lily on the GLUE benchmark are provided in Table 10. We fix the learning rate of both the backbone and the head at 5×10^{-3} and instead tune the scaling factor Lily_s, where Lily_s $\in \{0.01, 0.1, 1.0\}$. The rank r is fixed at 32, and the random seed is set to 0. The baseline results are taken from FourierFT (Gao et al., 2024b).

B.1.3 Visual Task Adaptation Benchmark

We provide the hyperparameters for Lily on the VTAB-1K benchmark in Table 11. Specifically, we fix the learning rate at 1×10^{-3} with a weight decay of 1×10^{-4}. For ViT, we tune the scaling factor Lily_s $\in \{0.01, 0.1, 1.0, 10.0\}$ to maximize performance, following Jie et al. (2023) and Jie and Deng (2023). For Vim, we fix Lily_s to 1.0. Additionally, we search for the hyperparameters *ne*_1 and *ne*_2 within the range {2, 3, 4}, as these numbers divide the number of layers in the ViT model (12 in ViT-B).

```python
class lily_adapter(nn.Module):
    """
    Implementation of a Lily adapter for an adaptable target. For symplicity, we assume that the
    ↪ number of hp expert is equal to the number of LPs.

    args:
        hidden_dim: hidden dimension
        ne: number of experts
        lp: low-dimensioanl projector
        hps: high-dimensioanl projector experts
        mlp: whether the adpatation target is located in MLP
    """
    def __init__(self, hidden_dim, ne, lp, hps, mlp=False):
        super().__init__()
        self.hps = hps
        self.ne = ne
        self.lp = lp
        self.router = nn.Linear(hidden_dim, ne, bias=False)
        if mlp:
            self.non_linear = nn.ReLU()
        else:
            self.non_linear = nn.Identity()
    def forward(self, x):
        hidden = self.non_linear(self.lp(x))
        router_logits = self.router(hidden) # [B, N, num_of_experts]
        router_probability = F.softmax(router_logits, dim=-1) # [B, N, ne]
        expert_probabilities = router_probability.mean(dim=(0, 1))
        combined_hp = torch.einsum("e,eio->io", expert_probabilities, self.hps)
        return torch.matmul(hidden, combined_hp)
```

Figure 7: Implementation of Lily in the VTAB-1K benchmark.

For Vim, we use the implementation discussed in Section A.1, which does not share As across layers. Therefore, *ne*_1 in this setting is fixed to the number of layers in Vim (22 in this case), while we search for *ne*_2 in {3, 6} and {5, 6, 17} separately for Lily-S and Lily-L. Note that *ne* is only set for the input projection in Vim. For the delta transformation, we use only a single B expert to reduce the parameter cost.

In the ViT experiments, the rank r is fixed at 16. Meanwhile, in Vim's setting, we tune the ranks r for the delta transformation module and the input projection module separately. We use $(4, 4)$ and $(4, 8)$ separately for Lily-S and Lily-L.

B.2 Datasets

B.2.1 Commonsense Reasoning

We provide a short description of each datasets used in commonsense reasoning experiments in Table 12.

B.2.2 Natural Language Understanding

We provide detailed information about datasets in the GLUE benchmark in Table 13.

B.2.3 Visual Adaptation Benchmark

We provide detailed information about all the tasks from VTAB-1K benchmark in Table 14.

C Does Sharing As Result in Inferior Performance?

As mentioned earlier, we adopted a strategy of sharing the A across most of our experiments, ensuring that the number of As and B experts is consistent. This approach offers two key benefits: simplicity and enhanced parameter efficiency. By sharing the A, we eliminate the need to set a separate A for each layer, thereby reducing the overall parameter count.

Our decision to share the A is based on the observation of overall redundancy among layers. Specifically, different layers have varying levels of importance (Zhang et al., 2023), and some less important layers do not require a dedicated A. By not setting a separate A for these layers, we avoid introducing unnecessary parameter overhead while maintaining negligible impact on performance. To test whether sharing A results in inferior performance, we conducted experiments without A sharing on the VTAB-1K benchmark. The results, shown in Table 15, indicate that the best overall performance (77.3%) is the same as in the A-sharing setting. This suggests that even when we employ one A for each layer, the performance gain is negligible, and many of the parameters are, in fact, redundant. However, not sharing As leads to additional

Table 6: Hyperparameter configuration from the MiLoRA paper.

MiLoRA hyperparameters	
Rank r	32
α of LoRA	64
α of PiSSA	32
Dropout	0.05
Optimizer	AdamW
LR	3e-4
LR Scheduler	Linear
Batch Size	16
Warmup Steps	100
Epochs	3
Placement	query, key, value, MA up, MA down

Table 7: Hyperparameter configuration from the PiSSA paper.

PiSSA hyperparameters	
α	Same as rank r
Dropout	0.0
Optimizer	AdamW
LR	2e-5
LR Scheduler	cosine
Batch Size	128
Warmup Ratio	0.03
Epochs	1
Placement	query, key, value, output, gate, MA up, MA down

Table 8: Hyperparameter configuration for LoRA using Falcon-Mamba as backbone.

LoRA hyperparameters	
Rank r	2
α	16
Dropout	0.05
Optimizer	AdamW
LR	3e-4
LR Scheduler	Linear
Batch Size	16
Epochs	1
Placement	input, delta

parameter overhead, which reduces the parameter efficiency of Lily. Therefore, A-sharing is an effective strategy to eliminate redundancy among As and enhance the parameter efficiency of Lily.

D Where to Apply Lily in Transformers?

E Performance Analysis on VTAB-1K Benchmark with Lily on Transformer Modules

PEFT methods have been predominantly explored on the Transformer architecture, which consists of multi-head self-attention (MHSA) and multi-layer perceptron (MA) as its core modules. In this section, we analyze the impact of fine-tuned modules on performance using Lily. Specifically, we compare Lily's performance on the VTAB-1K benchmark under four settings:

- Applying Lily solely to the query and value transformation module in MHSA (denoted as "qv").
- Applying Lily solely to the MA module (denoted as "mlp").
- Applying Lily to both the query and value transformation module in MHSA and the MA module (denoted as "qvmlp").

Table 9: Best Hyperparameter configuration for Lily using Falcon-Mamba and LLaMA3 as backbones.

	Falcon-Mamba	**LLaMA3**
Rank r	40	16
*ne*_1	4	4
*ne*_2	4	4
Dropout	0	0
Optimizer	AdamW	AdamW
LR	3e-4	3e-4
LR Scheduler	Linear	Linear
Batch Size	16	16
Epochs	1	3
Placement	input	query, key, value

Table 10: Hyperparameter of Lily on GLUE benchmark.

Hyperparameter	STS-B	RTE	MRPC	CoLA	SST-2	QNLI	MNLI	QQP
Optimizer	AdamW							
LR Schedule	Linear							
Learning Rate (Lily)	5e-3							
Learning Rate (Head)	5e-3							
Max Seq. Len	512	512	512	512	512	512	512	512
Lily_s	0.1	0.1	0.1	0.01	0.01	0.01	0	0
*ne*_1	2	3	2	4	2	2	0	0
*ne*_2	2	3	2	4	2	2	0	0
Batch Size	64	32	50	64	32	32	0	0

- Applying Lily to both the key and value transformation module in MHSA and the MA module (denoted as "kvmlp").

To ensure a fair comparison, we tune the hyperparameters to maintain a similar parameter count across all settings. Additionally, to further investigate whether sharing the low-rank projection (A) affects performance, we do not share A in this experiment.

The results are presented in Table 15. We observe that the "kvmlp" setting achieves the best performance, with an average accuracy of 77.3%. In contrast, adapting only the MHSA module ("qv") yields the worst performance. Furthermore, we note that adapting both the MHSA and MA modules (qvmlp and kvmlp) generally leads to superior results compared to adapting only one specific module (qv and mlp). This suggests that both MA and MHSA play crucial roles in overall model performance, and adapting both is essential for effective adaptation.

Notably, even when applying Lily solely to the MHSA module, which results in the worst performance among the four settings (76.9%), it still outperforms LoRA by a significant margin (0.5%). This underscores the efficiency of Lily, as it uses fewer parameters than LoRA, even without A sharing.

F Where to Apply Lily in Mamba?

Nearly all previous PEFT method studies have focused on Transformers, while Mamba is a relatively new architecture, and therefore, there has been little research on PEFT methods for Mamba. In this section, we briefly analyze the pros and cons of adapting Mamba's modules. A Mamba block consists of regular linear projection layers and a core component, the SSM module (Gu and Dao, 2023), (Zhu et al., 2024). Specifically, in the SSM module, Mamba utilizes parameters (Δ, A, B, C) to transform an input sequence $x(t)$ into an output sequence $y(t)$ using a hidden state $h(t)$. The discretization process converts A and B into $\bar{A}$ and $\bar{B}$, respectively, using the time step size parameter Δ. Structured state space models, inspired by continuous systems, can be computed similarly to

Table 11: Hyperparameter configuration for Lily on VTAB-1K benchmark.

	Vision Transformer	**Vision Mamba**
Optimizer	AdamW	AdamW
Batch Size	64	64
Learning Rate	1E-3	1E-2
Weight Decay	1E-4	1E-3
# Epochs	100	100
LR Decay	cosine	cosine

Table 12: Details of the datasets used in our commonsense reasoning tasks.

Benchmark	**Description**	**# Test Questions**
ARC-c	Multiple-choice science	2376
ARC-e	Multiple-choice science	1172
OBQA	Multi-step reasoning	500
SIQA	Social implications	1954
WinoG	Fill-in-a-blank	1267
PIQA	Physical commonsense	1830
BoolQ	Yes/no questions	3270
HellaS	Commonsense NLI	10042

Table 13: Information about datasets in the GLUE benchmark, with STS-B being a regression task and all other tasks falling into the categories of single-sentence or sentence-pair classification.

Corpus	**Metrics**	**Task**	**# Train**	**# Val**	**# Test**	**# Labels**
Single-Sentence Tasks						
CoLA	Matthews Corr.	Acceptability	8.55k	1.04k	1.06k	2
SST-2	Accuracy	Sentiment	67.3k	872	1.82k	2
Similarity and Paraphrase Tasks						
MRPC	Accuracy/F1	Paraphrase	3.67k	408	1.73k	2
STS-B	Pearson/Spearman Corr.	Sentence similarity	5.75k	1.5k	1.38k	1
QQP	Accuracy/F1	Paraphrase	364k	40.4k	391k	2
Inference Tasks						
MNLI	Accuracy	NLI	393k	19.65k	19.65k	3
QNLI	Accuracy	QA/NLI	105k	5.46k	5.46k	2
RTE	Accuracy	NLI	2.49k	277	3k	2

RNNs or in the form of global convolution due to their linear time invariance (LTI) property. Mamba introduces a selective property to the structured state space model, tying parameters to the current input. This breaks the LTI property and hinders parallel training. To address this, Mamba employs a hardware-aware algorithm, enabling its SSM module to possess the selective property while performing parallel training.

To be specific, the discretization process can be expressed as:

$$\begin{aligned} \bar{A} &= \exp(\Delta A) \\ \bar{B} &= (\Delta A)^{-1}(\exp(\Delta A) - I) \cdot \Delta B \end{aligned} \tag{6}$$

After that, the calculation in Mamba can be expressed as:

$$\begin{aligned} h_t &= \bar{A} h_{t-1} + \bar{B} x_t \\ y_t &= C h_t \end{aligned} \tag{7}$$

where h_t is the hidden state at time t and x_t is

Table 14: Detailed information about the datasets in VTAB-1K benchmark.

	Dataset	Train	Val	Test	#Classes
VTAB-1k	CIFAR100	800/1,000	200	10,000	100
	Caltech101			6,084	102
	DTD			1,880	47
	Oxford-Flowers102			6,149	102
	Oxford-Pets			3,669	37
	SVHN			26,032	10
	Sun397			21,750	397
	Patch Camelyon			32,768	2
	EuroSAT			5,400	10
	Resisc45			6,300	45
	Retinopathy			42,670	5
	Clevr/count			15,000	8
	Clevr/distance			15,000	6
	DMLab			22,735	6
	KITTI-Dist			711	4
	dSprites/location			73,728	16
	dSprites/orientation			73,728	16
	SmallNORB/azimuth			12,150	18
	SmallNORB/elevation			12,150	18

Table 15: Performance on VTAB-1K benchmark when applying Lily to various modules in Transformer. The implementation here does not share A for simplicity (i.e., each layer has one A).

		Natural							**Specialized**				**Structured**							
	Average	**Cifar100**	**Caltech101**	**DTD**	**Flowers102**	**Pets**	**SVHN**	**Sun397**	**Camelyon**	**EuroSAT**	**Resisc45**	**Retinopathy**	**Clevr-Count**	**Clevr-Dist**	**DMLab**	**KITTI-Dist**	**dSpr-Loc**	**dSpr-Ori**	**sNORB-Azim**	**sNORB-Ele**
qv	76.9	73.2	92.3	72.2	99.3	91.4	89.0	56.5	87.6	95.2	84.8	75.9	83.7	65.8	52.8	81.2	87.6	52.4	36.3	43.4
mlp	77.0	74.0	92.6	72.2	99.4	91.5	89.0	55.9	88.2	95.5	85.4	76.0	83.3	63.1	53.0	81.4	86.5	53.8	35.6	43.3
qvmlp	77.1	73.9	93.2	72.7	99.4	91.6	89.7	56.5	87.9	95.3	85.0	76.1	84.6	65.2	53.0	82.1	86.7	53.0	36.0	42.8
kvmlp	77.3	74.0	92.3	72.6	99.3	91.5	89.2	56.7	88.2	95.4	85.3	76.0	84.6	64.9	53.4	81.7	87.5	52.9	36.9	45.2

the corresponding input token. Delta projection is a module in SSM that's learnable and tasked with transforming the parameter Δ. Since adapting the delta projection alone can indirectly adapt the entire SSM module (i.e., $\bar{A}$ and $\bar{B}$ are determined by Δ), it is the most critical component of the SSM module.

We investigate the performance of two adaptation strategies: adapting only the input linear projection layer (denoted as "in") and adapting both the input linear projection layer and the SSM (denoted as "Δ + in" since we only adapt the delta projection in the SSM module). Our results, as shown in Table 1, indicate that applying Lily solely to the input projection yields better performance than applying it to both the input and delta projection modules. This suggests that when adapting Mamba-based models under the paradigm of low-rank adaptation, it is optimal to adapt only the input projection module outside the SSM module. These findings highlight the need for further research into the impact of fine-tuned modules in Mamba on overall performance. Additionally, developing PEFT methods specifically tailored to Mamba-based models, whether for vision or language foundation models, is also a promising direction for future work.

G Performance with Different Learning Rates

Since we only tuned the learning rate in the commonsense reasoning experiment, we provide the performance of commonsense reasoning under different learning rates in Table 16.

Table 16: Commonsense reasoning results of Lily under various leanring rates.

Model	Lr	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
LLaMA3-8B	1e-3	70.7	84.6	77.6	87.8	77.3	88.5	74.1	80.8	80.2
	5e-4	71.8	86.5	77.9	82.8	83.1	88.6	76.8	81.4	81.1
	3e-4	72.9	85.6	77.8	92.7	83.3	89.7	77.6	82.8	82.8

H Does Selectivity Help?

Lily introduced selective weight combination to selectively incorporate information from other layers. To verify the effectiveness of this selectivity, we remove the router from Lily and evaluate the impact. The modified algorithm without the router is presented in Fig. 8. We conduct experiments on commonsense reasoning to investigate the effect of removing selectivity from Lily.

As shown in Table 17, removing selectivity from Lily results in generally poorer performance compared to vanilla Lily. This is likely because the lack of selectivity causes Lily to simply aggregate all the B expert, leading to inferior performance. This validates the design choice of using routers in Lily to selectively allocate weights to B experts, rather than simply summing them.

I How to Allocate Parameters?

Since Lily alters the traditional LoRA's layer-bound setup, increasing the parameters of Lily can be achieved through two approaches: 1) increasing *ne*, i.e., increasing the number of A and B experts, and 2) increasing the rank, i.e., increasing the parameter size of each individual A or B expert. In this section, we investigate which factor has the greatest impact on performance. We conduct experiments on the commonsense reasoning task. Specifically, we maintain the same parameter count and learning rate, and achieve the same parameter count by setting different ranks and adjusting the corresponding *ne* (e.g., r=16, *ne*=4 versus r=8, *ne*=8). The results are shown in Fig. 9, from which we observe that more A and B experts with smaller rank (i.e., bigger *ne* and smaller rank) generally performs worse. We argue that this is because, although increasing the attention granularity allows for finer details, the resulting performance gain is not as significant as the gain obtained by increasing the rank, i.e., increasing the model's capacity to learn more information. This gives us an insight that, in Lily, increasing *ne* to increase the parameters is less effective than directly increasing the rank in terms of potential performance gain.

```python
class lily_adapter_monoscale(nn.Module):

    def __init__(self, hidden_dim, ne, lp, hps, mlp=False):
        super().__init__()
        self.hps = hps
        self.ne = ne
        self.lp = lp
        self.scale = 1 / ne
        if mlp:
            self.non_linear = nn.ReLU()
        else:
            self.non_linear = nn.Identity()
    def forward(self, x):
        hidden = self.non_linear(self.lp(x))
        combined_hp = torch.sum(self.hps, 0) * self.scale
        return torch.matmul(hidden, combined_hp)
```

Figure 8: Implementation of Lily with no selectivity.

Table 17: Commonsense reasoning results of Lily without selectivity. We provide results using two learning rates.

Model	Lr	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
	3e-4	64.0	82.6	78.5	77.0	79.6	88.4	74.5	82.0	78.3
LLaMA3-8B	5e-4	71.3	85.5	78.1	84.3	79.6	86.4	76.1	79.0	79.8

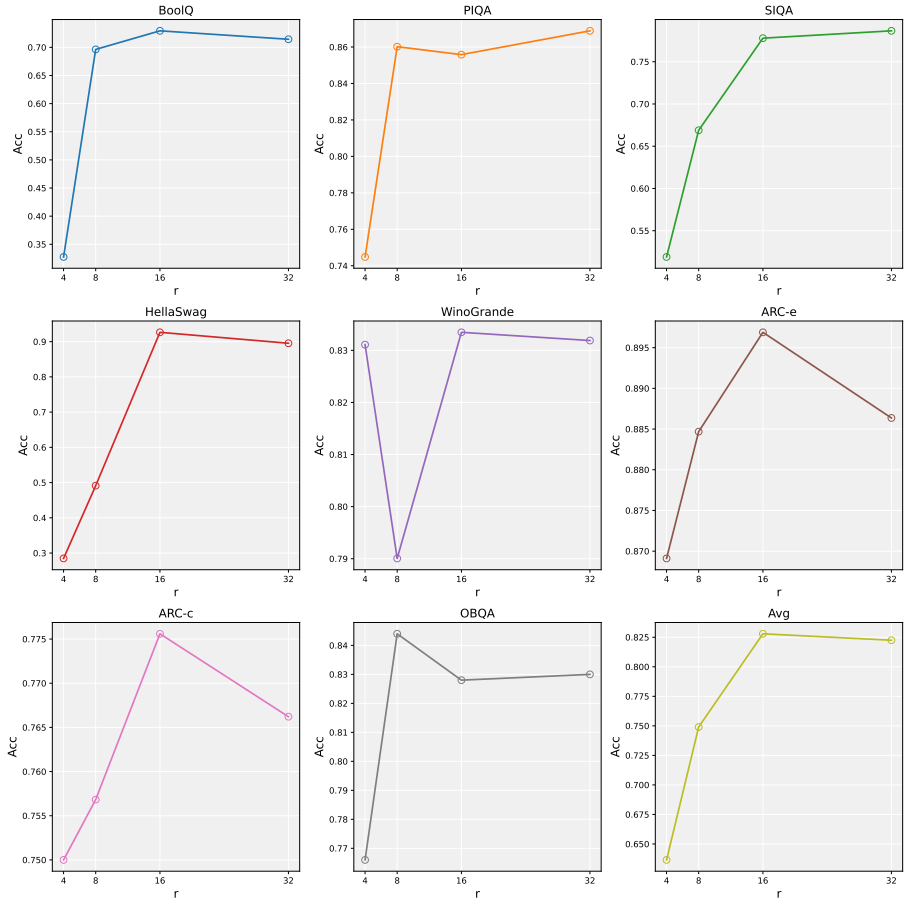

Figure 9: Results on commonsense reasoning tasks when applying different settings of rank. The hyperparameter *ne* is specifically tuned to maintain the same amount of parameter count for a fair comparison.

J More on Subject-driven Generation

We provide more results on subject-driven generation in Fig. 10 and Fig. 11.

K From a Feature Merging Perspective

Apart from having higher-rank weight updates than LoRA, Lily also enables comprehensive information access across layers. Lily enables access to information or features from all other layers when adapting a target module at a specific layer thanks to the inter-connectivity of the adapters. We aim to understand how Lily achieves this comprehensive information access from the perspective of visual tasks as shown in Fig. 12. We can observe that, in Lily, the distinctness of the attention maps between layers is not as pronounced as in LoRA. This validates Lily's ability to enable all-level information access, since adaptation at each layer takes into account features from other layers. Additionally, we specifically visualize the actual feature differences between different layers in Fig. 13. We observe that Lily has more points with low feature differences (blue color) than LoRA, indicating that the distinctness of features between layers in Lily is generally lower than in LoRA. This further demonstrates Lily's ability to enable comprehensive information access. Although we enable all-level information access, what prevents the features from becoming completely identical is the selectivity introduced by Lily, which we specify in the following section.

L More on Attention Maps of Lily and LoRA

We provide more visualization results of the attention map from both LoRA and Lily on Caltech101 dataset from VTAB-1K benchmark in Fig. 14.

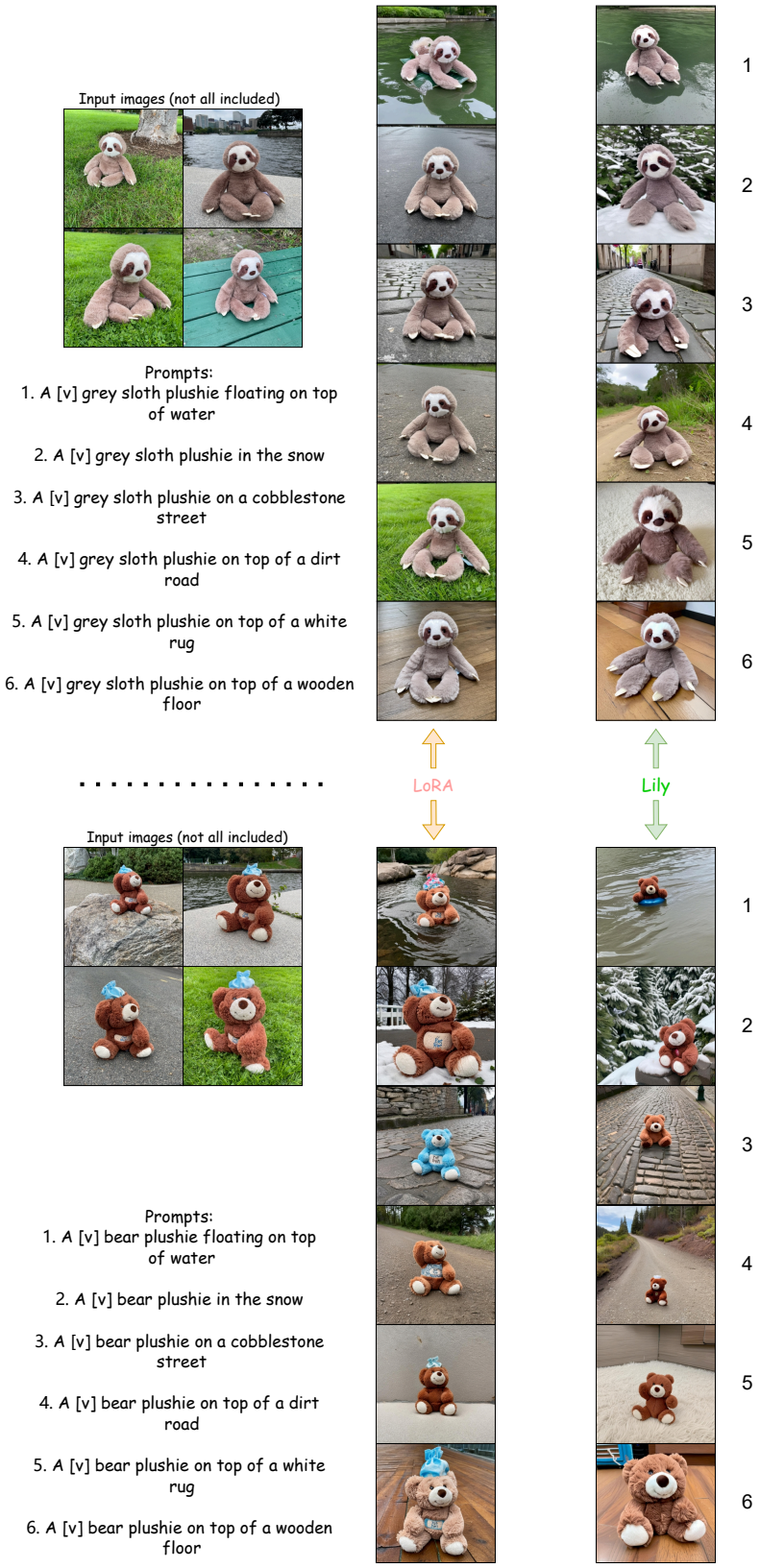

Figure 10: More subject-driven generation results for unreported subjects.

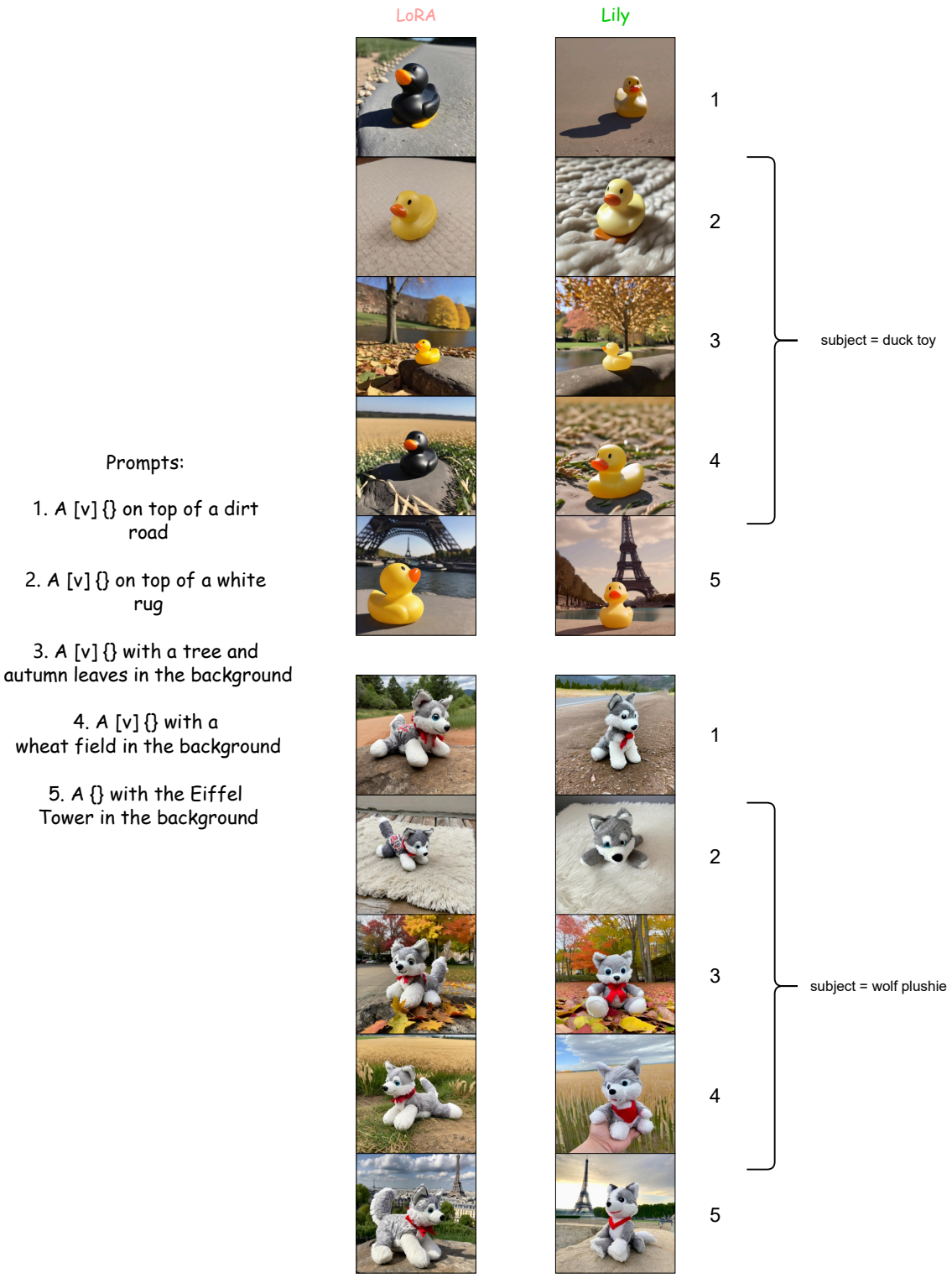

Figure 11: More subject-driven generation results for subjects that are reported in the experiment section.

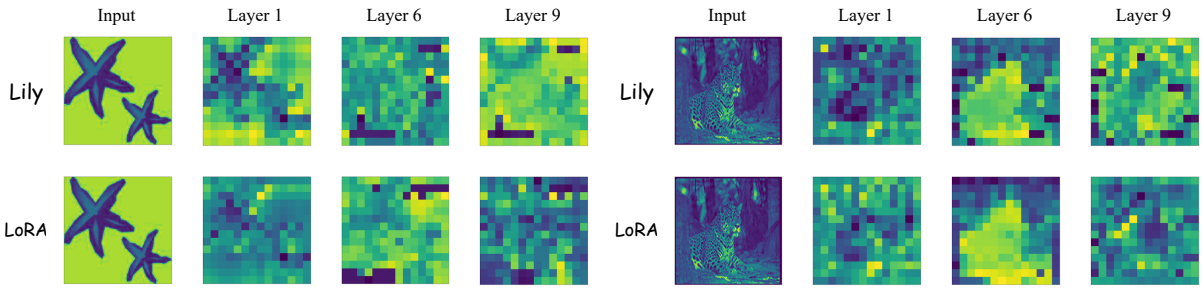

Figure 12: Attention maps of Lily and LoRA. The input images for the example here are taken from Caltech101 datasets from VTAB-1K benchmark. It can be observed that features from a certain layer have more similarity to those in other layers in Lily than in LoRA.

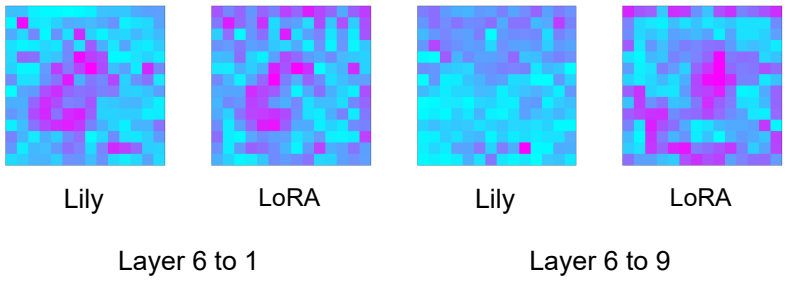

Figure 13: Feature difference measured in absolute distance for each element. We compare Lily and LoRA in terms of the difference between features from different layers. In this example image taken from Caltech101, we visualize the feature difference between layers 6 and 1, as well as between layers 6 and 9.

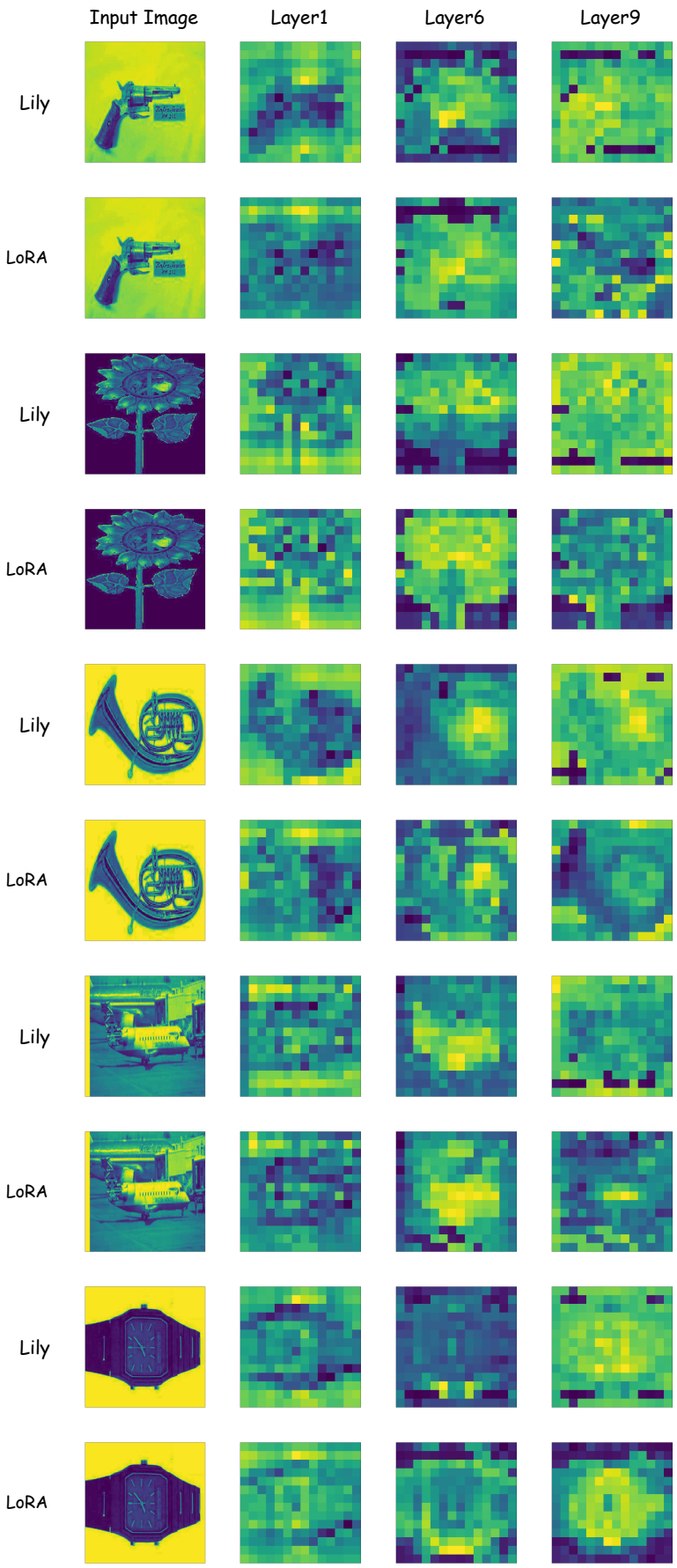

Figure 14: More results of attention maps from LoRA and Lily. All images are taken from Caltech101 dataset.