Supplementary Materials for VL-ADAPTER: Parameter-Efficient Transfer Learning for Vision-and-Language Tasks

Yi-Lin Sung Jaemin Cho Mohit Bansal
UNC Chapel Hill
{ylsung,jmincho,mbansal}@cs.unc.edu

A. Data Statistics

Data statistics for the dataset we used are listed in Table 2.

B. Training and Evaluation

We perform an extensive hyper-parameter search for our models. See Appendix E for details. We use AdamW to train the model, unless we additionally specify and apply a linear decay scheduler. We train the models for 20/7 epochs for image-text/video-text tasks, and warm up the learning rate from 0 to the highest learning rate in the first 2 epochs. In the image-text experiments, we use batch size 500 for CLIP-BART and 250 for CLIP-T5, and the total training time is about 20 hours and 40 hours for CLIP-BART and CLIP-T5 with one A6000 GPU (48G memory), respectively. In video-text experiments, we use batch size 50 for CLIP-BART and the total training time is about 10 hours. We select the last checkpoint for evaluation and report the evaluation score of the four tasks as well as the average score in our experiments. The percentage of updated parameters is also reported as the metric for approaches' efficiency, and we do not take visual encoder into account for computation since it is frozen.

C. Details for Prompt-tuning

Prompt-tuning [7] adds trainable parameters to the encoder's inputs for adapting those parameters for new tasks without changing the model. Specifically, we assume the input indices for generating prompts are $1, 2, ..., N_p \in \mathbb{N}$, where N_p is the length of prompts. We next apply a three-layer neural network to transform the prompts embeddings to the correct dimension for the language model. The first layer is an embedding layer, parameterized by $\theta_E \in \mathbb{R}^{N_p \times d_i}$, and the rest of the two layers are parameterized by $\theta_D \in \mathbb{R}^{d_i \times d}$ and $\theta_U \in \mathbb{R}^{d \times d_i}$. Since the architecture of the prompt network is quite similar to the adapter module, we use the same notations as we used in adapters for simplicity. The mathematic form can be written as the following,

$$\begin{aligned} h &= f_{\theta_E}(p) \\ h_p &= f_{\theta^U}(\sigma(f_{\theta^D}(h))) \end{aligned} \tag{1}$$

where $p \in 1, 2, ..., N_p$, h_p being the prompt of index p, and we use Tanh as the activation function. Next, we can combine the prompt embeddings with vision and sentence embedding, feed-forwarding to the model, and train them with backpropagation. The trainable parameters consist of the input prompts embeddings and the parameters of the three-layer neural network. Note that unlike in adapter modules that d is smaller than d_i for saving memory, d in the prompt network is sometimes greater than d_i since it is the main hyper-parameter to increase the number of trainable parameters. The length of the prompt N_p does not contribute much to the number of parameters since it only influences the embedding layer, which usually is a small layer. Thus, using longer prompts is a parameter-efficient method to train models. However, the memory usage would increase significantly with longer prompts due to the quadratic cost of attention layer on input lengths. For a fair comparison, we maximize N_p to use the same amount of memory as being used in adapter-based approaches (around 40 GB).

D. Details for LoRA

Assume the initial weight for a layer is $\theta^{d_i \times d_o}$, LoRA [3] learns two low-rank matrices $A \in R^{d_i \times d}$ and $B \in R^{d \times d_o}$ ($d \ll d_i, d_o$) to approximate the weight's updates, that is

$$\Delta\theta = AB \tag{2}$$

The output of this layer can be written as $f_{\theta+AB}(h)$. Hu et al. [3] apply this trick to attention layers (not in feed-forward layers), and they also update bias terms of the model. Compared to the original model, using adapters or prompt-tuning, which modify the network or inputs, causes

Method	Best Learning Rate	Updated Params (%)	VQA Karpathy test Acc. (%)	GQA test-dev Acc. (%)	NLVR^2 test-P Acc. (%)	COCO Cap. Karpathy test CIDEr	Avg.
(A) Full fine-tuning	1×10^{-4}	100.00	67.6	56.7	73.0	112.9	**77.6**
(B) Multiple Adapters							
(B.1) - $\boldsymbol{d = 96}$	3×10^{-4}	12.22	65.4	54.0	69.8	114.3	**75.9**
(B.2) - $d = 48$	1×10^{-3}	7.58	65.4	53.7	65.3	115.0	74.9
(C) Half-shared Adapters ($d = 96$)							
(C.1) - sharing downsampling layers	3×10^{-4}	8.40	65.2	53.3	70.2	113.8	75.6
(C.2) - **sharing upsampling layers**	3×10^{-4}	8.36	65.2	53.4	71.2	113.7	**75.9**
(D) Single Adapter							
(D.1) - $d = 192$	1×10^{-3}	7.54	66.5	54.0	73.5	115.8	77.4
(D.2) - $\boldsymbol{d = 96}$	1×10^{-3}	4.36	65.9	54.5	74.2	114.9	**77.4**
(D.3) - $d = 64$	1×10^{-3}	3.30	65.2	53.8	72.3	114.5	76.4
(D.4) - $d = 48$	1×10^{-3}	2.78	64.7	53.9	71.5	114.2	76.1
(D.5) - $d = 24$	1×10^{-3}	1.98	63.5	52.2	71.0	113.5	75.1
(F) Hyperformer							
(F.1) - $\boldsymbol{d = 96, d_p = 8}$	1×10^{-3}	5.79	65.1	53.4	72.3	114.6	**76.4**
(F.2) - $d = 96, d_p = 4$ *	1×10^{-3}	3.87	65.0	53.2	51.1	114.9	71.0
(F.3) - $d = 48, d_p = 8$	1×10^{-3}	3.77	64.5	52.5	71.3	114.3	75.7
(G) Multiple Compacters ($d = 48$)							
(G.1) - w/ sharing weights, w/ low-rank param. ($r = 1$), $k = 1$	1×10^{-3}	1.381	50.8	41.6	53.5	104.9	62.7
(G.2) - w/ sharing weights, w/ low-rank param. ($r = 1$), $k = 4$	1×10^{-3}	1.381	52.6	43.5	54.0	111.6	65.4
(G.3) - w/ sharing weights, w/ low-rank param. ($r = 1$), $k = 8$	1×10^{-3}	1.382	52.2	42.4	58.3	109.8	65.7
(G.4) - w/ sharing weights, w/ low-rank param. ($r = 1$), $k = 12$	1×10^{-3}	1.383	53.9	43.7	60.4	111.1	67.3
(G.5) - w/ sharing weights, w/o low-rank param., $k = 4$	1×10^{-3}	2.83	52.7	42.7	59.7	112.2	66.8
(G.6) - w/o sharing weights, w/o low-rank param., $k = 2$	1×10^{-3}	4.42	64.0	52.9	68.3	115.7	75.2
(G.7) - w/o sharing weights, w/o low-rank param., $k = 4$	1×10^{-3}	2.84	62.4	51.4	68.6	115.5	74.5
(G.8) - w/o sharing weights, w/o low-rank param., $k = 8$	1×10^{-3}	2.11	61.4	50.9	68.9	115.4	74.1
(H) Multiple Compacters ($d = 96$)							
(H.1) - **w/o sharing weights, w/o low-rank param., $\boldsymbol{k = 2}$**	1×10^{-3}	7.02	64.6	53.4	69.1	116.0	**75.8**
(I) Single Compacter ($d = 96$)							
(I.1) - **w/o sharing weights, w/o low-rank param., $\boldsymbol{k = 2}$**	1×10^{-3}	2.67	64.2	53.3	71.7	114.1	**75.8**
Single Compacter ($d = 48$)							
(I.2) - w/o sharing weights, w/o low-rank param., $k = 2$	1×10^{-3}	1.59	61.6	50.7	69.0	114.0	73.8
(J) Multiple Prompts							
(J.1) - $\boldsymbol{N_p = 40, d_m = 800}$	1×10^{-3}	4.53	43.8	38.1	51.1	104.6	59.4
(J.2) - $N_p = 40, d_m = 100$	1×10^{-3}	1.64	47.4	37.0	49.8	108.6	**60.7**
(K) Single Prompt							
(K.1) - $\boldsymbol{N_p = 40, d_m = 800}$	1×10^{-3}	2.00	44.0	36.3	51.8	103.9	**59.0**
(K.2) - $N_p = 40, d_m = 100$	1×10^{-3}	1.25	43.5	36.4	52.0	103.4	58.8

Table 1. The multi-task evaluation results for CLIP-BART on VQA, GQA, NLVR^2 and COCO Caption between adapter-based approaches with different hyper-parameters. We bold the highest average accuracy separately for each approach, and we also bold the best configuration we used in the main paper. Note that we don't use V&L pre-training for every model. * denotes the NLVR results might be improved if we use different learning rates.

Type	Dataset	Data size (# videos / # QA pairs, # captions) Train	Validation	Test
Image	VQA [2]	113.2K/605.1K	5.0K/26.7K	5.0K/26.3K
	GQA [4]	72.1K/943.0K	10.2K/132.1K	398/12.6K
	NLVR^2 [10]	103.2K/86.4K	8.1K/7.0K	8.1K/7.0K
	COCO Cap. [1]	113.2K/566.8K	5.0K/5.0K	5.0K/5.0K
Video	TVQA [5]	17.4K/122.0K	2.2K/15.3K	2.2K/15.3K
	How2QA [8]	24.5K/34.2K	3.1K/3.1K	3.1K/3.1K
	TVC [6]	17.4K/86.7K	10.8K/43.6K	10.8K/43.6K
	YC2C [11]	10.3K/10.3K	3.5K/3.5K	1.6K/1.6K

Table 2. The statistics of the datasets used in our experiments.

extra computation in the inference. However, there is no extra overhead in LoRA since we can add the updates back to the model after training.

E. Hyper-parameter Search

We search over the learning rates among $\{1 \times 10^{-4}, 3 \times 10^{-4}, 1 \times 10^{-3}\}$ for each hyper-parameter configuration. To reduce the cost of searching, we utilize a heuristic logic: we first search for the best learning rate for the one hyper-parameter configuration (randomly chosen) and then use the same learning rate for other configurations. We perform another learning rate search only if the results are diverged for some tasks (e.g. sometimes the results of NLVR^2 become very low at certain learning rates).

For the Adapter, the only hyper-parameter is the hidden dimension d. We also ablate two variants of Half-shared Adapters: sharing upsampling or downsampling layers. We include the search about the projected hidden dimension d_e for the task projector network in the Hyperformer. Regard-

Method	Best Learning Rate	Updated Params (%)	VQA Karpathy test Acc. (%)	GQA test-dev Acc. (%)	NLVR^2 test-P Acc. (%)	COCO Cap. Karpathy test CIDEr	Avg.
(A) Full fine-tuning	1×10^{-4}	100.00	67.3	56.5	75.4	113.1	**78.1**
(B) Multiple Adapters							
(B.1) - $\boldsymbol{d = 192}$	1×10^{-3}	24.56	66.0	55.7	51.8	111.9	71.3
(B.2) - $d = 96$	1×10^{-3}	14.29	66.1	55.7	52.5	112.8	**71.8**
(C) Single Adapter							
(C.1) - $d = 384$	3×10^{-4}	14.25	67.6	55.9	73.6	111.8	77.2
(C.2) - $\boldsymbol{d = 192}$	3×10^{-4}	7.98	67.6	56.2	73.9	111.8	**77.4**
(C.3) - $d = 96$	1×10^{-3}	4.49	66.4	55.5	72.7	111.5	76.5
(C.4) - $d = 48$	1×10^{-3}	2.64	65.7	54.7	70.9	111.1	75.6
(D) Hyperformer							
(D.1) - $\boldsymbol{d = 192, d_p = 8}$	1×10^{-3}	6.37	65.5	55.1	71.5	112.2	**76.1**
(D.2) - $d = 192, d_p = 4$	1×10^{-3}	3.99	65.0	53.9	70.4	111.7	75.2
(E) Multiple Compacters ($d = 192$)							
(E.1) - **w/o sharing weights, w/o low-rank param., $k = 2$** *	1×10^{-3}	14.30	66.1	55.0	52.1	112.9	71.5
(E.2) - w/o sharing weights, w/o low-rank param., $k = 4$ *	1×10^{-3}	8.06	65.4	55.0	52.2	113.2	**71.5**
(E.3) - w/o sharing weights, w/o low-rank param., $k = 8$ *	1×10^{-3}	4.66	63.3	52.9	51.7	110.4	69.6
(F) Single Compacter ($d = 192$)							
(F.1) - **w/o sharing weights, w/o low-rank param., $k = 2$**	1×10^{-3}	4.49	67.0	56.6	72.5	112.7	**77.2**
(F.2) - w/o sharing weights, w/o low-rank param., $k = 4$	1×10^{-3}	2.65	66.1	55.2	71.8	111.7	76.2
(F.3) - w/o sharing weights, w/o low-rank param., $k = 8$	1×10^{-3}	1.72	65.2	54.1	71.6	111.5	75.6

Table 3. The multi-task evaluation results for CLIP-T5 on VQA, GQA, NLVR^2 and COCO Caption between adapter-based approaches with different hyper-parameters. We bold the highest average accuracy separately for each approach, and we also bold the best configuration we used in the main paper. Note that we don't use V&L pre-training for every model. * denotes the NLVR^2 results might be improved if we use different learning rates.

Model	Approach	Learning Rate	Batch size	Other hyper-parameters
CLIP-BART	Full fine-tuning	1×10^{-4}	500	-
	Multiple Adapters	3×10^{-4}	500	$d = 96$
	Half-shared Adapters	3×10^{-4}	500	sharing upsampling layers, $d = 96$
	Single Adapter	1×10^{-3}	500	$d = 96$
	Hyperformer	1×10^{-3}	500	$d = 96, d_p = 8$
	Multiple Compacters	1×10^{-3}	500	remove share weight and low-rank, $d = 96, k = 2$
	Single Compacter	1×10^{-3}	500	remove share weight and low-rank, $d = 96, k = 2$
	Multiple Prompts	1×10^{-3}	500	$N_p = 40, d_m = 800$
	Single Prompt	1×10^{-3}	500	$N_p = 40, d_m = 800$
CLIP-T5	Full fine-tuning	1×10^{-4}	250	-
	Multiple Adapters	1×10^{-3}	250	$d = 192$
	Single Adapter	3×10^{-4}	250	$d = 192$
	Hyperformer	1×10^{-3}	250	$d = 192, d_p = 8$
	Multiple Compacters	1×10^{-3}	250	remove share weight and low-rank, $d = 192, k = 2$
	Single Compacter	1×10^{-3}	250	remove share weight and low-rank, $d = 192, k = 2$

Table 4. The best hyperparameter configurations for different parameter-efficient training approaches.

ing the Compacter, we have tried different numbers of Kronecker products (k), hidden dimension d, and whether sharing weights and using low-rank parameterization. We also tune the d_m for prompt-tuning.

E.1. CLIP-BART Hyper-parameter Search

We show the results of the hyper-parameter search in Table 1. We bold the final configurations used in the main paper and we also list the configurations in Table 4. The exception is that we use the same hyper-parameters for the "Single" and "Multiple" approaches. For example, even though Multiple Prompts perform better when $d_m = 100$, we still use $d_m = 800$ for both Multiple Prompts and Single Prompt for consistency (J and K rows in Table 1).

E.2. CLIP-T5 Hyper-parameter Search

We display the results of the hyper-parameter search for CLIP-T5 in Table 3 and final configurations for each method in Table 4. We find that the Compacter (F.1 in Table 3) shows the different fashion in T5: it can perform similarly to the Single Adapter (C.2 in Table 3) using fewer parameters. This might because the Compacter is mainly

Method	VQA	GQA	NLVR^2	COCO Cap.	Avg.
CLIP-BART					
- w/o prompt	66.7	56.5	**73.2**	112.4	77.2
- w/ prompt	**67.6**	**56.7**	73.0	**112.9**	**77.6**
CLIP-BART + Single Adapter					
- w/o prompt	65.1	53.9	72.7	**115.6**	76.8
- w/ prompt	**65.9**	**54.5**	**74.2**	114.9	**77.4**

Table 5. Ablation results of adding task-specific prompts.

Method	Updated Params (%)	VQA test-std	GQA test-std
CLIP-BART			
+ Full fine-tuning	100.00	70.1	52.5
+ Single Adapter	4.18	68.3	50.9
+ Single LoRA	5.93	67.3	50.0
+ Single Prompt	2.00	45.3	37.3

Table 6. Leaderboard results of test-std split for representative approaches from different method families.

validated on T5 in [9].

F. Additional Experimental Results

Different Visual Representations. We compare the results (Table 1) of CLIP-BART and VL-BART (w/o pre-training) and find out that there is a small improvement in using CLIP features over R-CNN features (77.6 vs. 76.7). Note that our CLIP also uses images with a smaller size (224×224 vs. 800×1333), so the result proves the effectiveness of pre-trained cross-modality features.

Adapters with Task-specific Prompts. We experiment to remove task-specific prompts before the input sequence, namely, from "[task]: [input]" to "[input]", where [task] is task indicator, such as "vqa", "gqa", "nlvr", and "caption". The ablation is only for the approaches using one set of parameters for multi-tasking, such as full fine-tuning and Single Adapter. We exclude Hyperformer in this experiment since we follow the original implementation to remove all prompts and use task embedding as the condition. The results of whether to use task-specific prompts are displayed in Table 5. We find that using prompts can improve performance, and the improvement likely comes from resolving the confusion between tasks. However, the model still performs well without prompts. We hypothesize that the data distribution between tasks is large enough for the model to understand to treat them differently, so the added prompts might become redundant. For example, there is no text input in MSCOCO, while there are two input images in NLVR^2.

Methods	VQA
Full fine-tuning	69.9
Multiple Adapters	66.7
Half-shared Adapters	66.6
Single Adapter	68.1
Hyperformer	67.5
Multiple Compacters	66.5
Single Compacter	66.4
Multiple LoRA	67.4
Single LoRA	67.0
Multiple Prompts	48.8
Single Prompt	45.4

Table 7. Test-dev results for VQA.

G. Leaderboard Results for VQA

While we use the Karpathy split for VQA evaluation in the main content, we also report the test-std results for representative approaches and test-dev results for all approaches in Table 6 and Table 7. In short, the trend remains similar as using the Karpathy split, and the Single Adapter still performs the best among parameter-efficient training methods.

H. Limitations

Next, we discuss some limitations of this work. We have carried out extensive experiments on four V&L tasks with our proposed CLIP-BART and CLIP-T5. However, different architectures have their own best hyper-parameters, and data distributions are varied across tasks, so our results and findings do not always guarantee to be generalized to other tasks. Furthermore, we experiment with the three popular adapter variants, but they cannot represent all the adapter-based approaches.

References

[1] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO captions: Data collection and evaluation server. *CoRR*, abs/1504.00325, 2015. 2

[2] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6325–6334, 2017. 2

[3] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685, 2021. 1

[4] Drew A Hudson and Christopher D Manning. Gqa: A new dataset for real-world visual reasoning and compositional

question answering. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[5] Jie Lei, Licheng Yu, Mohit Bansal, and Tamara L. Berg. Tvqa: Localized, compositional video question answering. In *EMNLP*, 2018. 2

[6] Jie Lei, Licheng Yu, Tamara L. Berg, and Mohit Bansal. Tvr: A large-scale dataset for video-subtitle moment retrieval. *ECCV*, 2020. 2

[7] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *EMNLP*, 2021. 1

[8] Linjie Li, Yen-Chun Chen, Yu Cheng, Zhe Gan, Licheng Yu, and Jingjing Liu. Hero: Hierarchical encoder for video+language omni-representation pre-training. *EMNLP*, 2020. 2

[9] Rabeeh Karimi mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021. 4

[10] Alane Suhr, Stephanie Zhou, Ally Zhang, Iris Zhang, Huajun Bai, and Yoav Artzi. A corpus for reasoning about natural language grounded in photographs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6418–6428, Florence, Italy, July 2019. Association for Computational Linguistics. 2

[11] Luowei Zhou, Chenliang Xu, and Jason J. Corso. Towards automatic learning of procedures from web instructional videos. In *AAAI*, 2018. 2