Supplementary Material of AdaPrefix++: Integrating Adapters, Prefixes and Hypernetwork for Continual Learning

Sayanta Adhikari[1], Dupati Srikar Chandra[1], P. K. Srijith[1], Pankaj Wasnik[2], Naoyuki Oneo[2]
[1]Indian Institute of Technology Hyderabad, India [2]Sony Research India, India
{ai22mtech12005, ai20resch11004}@iith.ac.in, srijith@cse.iith.ac.in,
{Pankaj.Wasnik, naoyuki.onoe}@sony.com

In the supplementary section, we provide details on our approaches, AdaPrefix and AdaPrefix++. In Appendix A, we provide the inference algorithm for our approach for different scenarios (TIL, CIL). In Appendix B.2 and Appendix B.3, we provide details about different experimental choices we considered while performing the experiments. In Appendix B.1.1, we provide different transformations associated with a transformer block when adapters and prefixes are added to a transformer block. In Appendix C, Appendix D and Appendix E, we provide different experiments for the voices of the hyperparameter that we have taken. In Appendix F.1, we provide details related to computational overheads for our approach and per-task running time for our experiments. In Appendix F, we further provide other experiments of our approach, showing the robustness of our approach in multiple setups. Tab. 5, Tab. 7, Tab. 10 and Tab. 12 provides results of comparison between our approach and different baselines in task-incremental and class-incremental scenarios. Tab. 6, Tab. 8, Tab. 11, and Tab. 13 provides results for different task orders for our approach AdaPrefix++. These experiments show that our approach doesn't change much with a change in the task order, showing our approach's robustness towards task order.

A. Algorithms

In this section, we provide the pseudo-code for the Inference algorithm of our approach *AdaPrefix++*. Algorithm 1 provides two inference algorithms, the first for TIL Inference and the second for CIL Inference. The inference algorithm for DIL is similar to CIL. All the notations used in the algorithm are the same as presented in the main paper.

Algorithm 1 *AdaPrefix++* Inference Algorithm

```
1:  procedure INFERENCE TIL(Trained Model $\hat{F}$, Input $(X_*^t, \mathcal{T}^t)$)        ▷ Task Incremental Inference
2:      Get prefix parameters using $P_i^{At} \leftarrow \mathcal{H}([e^{At}, l_i^A]; \theta_h)$ where $A \in \{K, V\}$.
3:      Get adapter weights, $\phi^t$.
4:      $\mathrm{p}_*^{\mathrm{t}} \leftarrow \hat{F}(X_*^t; \Theta_p, \theta_h, \phi^t, \theta_{cls}^t, \mathrm{L})$
5:      $\arg\max_{c\in\{1,\cdots,C\}} \mathrm{p}_*^t[c]$ is the predicted class.
6:  end procedure
7:  procedure INFERENCE CIL(Trained Model $\hat{F}$, Input $(X_*)$)        ▷ Class Incremental Inference
8:      Logit_List = [ ]
9:      for $t \leftarrow 1$ to $T$ do
10:         Get prefix parameters using $P_i^{At} \leftarrow \mathcal{H}([e^{At}, l_i^A]; \theta_h)$ where $A \in \{K, V\}$.
11:         Get adapter weights, $\phi^t$.
12:         $\mathrm{p}_*^{\mathrm{t}} \leftarrow \hat{F}(X_*; \Theta_p, \theta_h, \phi^t, \theta_{cls}^t, \mathrm{L})$
13:         Logit_List.append($\mathrm{p}_*^{\mathrm{t}}$)
14:     end for
15:     $\hat{t} \leftarrow \arg\min_{k\in[T]} \mathbb{E}$(Logit_List[$k$])        ▷ Entropy, $\mathbb{E}(p) = -\sum_i p_i \log(p_i)$
16:     $\arg\max_{c\in\{1,\cdots,C\}} \mathrm{p}_*^{\hat{t}}[c]$ is the predicted class.        ▷ $\mathrm{p}_*^{\hat{t}} \leftarrow$ Logit_List[$\hat{t}$]
17: end procedure
```

B. Implementation Details

In this section, we provide implementation details related to the different backbones used, the transformation associated with a standard transformer block when adapters and prefixes are added to it, and all our architectural choices. In this section, we also provide details related to the baselines used and methods used to compare them in the case of the TIL scenario. In this section, we also provide details related to optimization.

B.1. Backbone PLMs

For our study, we have considered various sizes of backbone PLMs in the domain of large pre-trained vision models. We have considered ViT-Large (ViT-L), ViT-Base (ViT-B), DeiT-Small (DeiT-S), and DeiT-Tiny (DeiT-T). Details of these models are provided in the Tab. 1

Table 1. Backbone PLM

Backbone PLM	**#Parameters**	**#Layers**	**#Heads**	**embed_dim**
ViT-L	307M	24	16	1024
ViT-B	85M	12	12	768
DeiT-S	21M	12	6	384
DeiT-T	5M	12	3	192

B.1.1 Transformations in the Transformer Block of PLMs

Let us consider the transformations in the i^{th} layer (among L layers) for task t (among T tasks) in a transformer block of the proposed *AdaPrefix* approach. The transformer block mainly has two parts: a MHA and a FFN Layer. Our approach changes the MHA using Prefixes and changes the FFN using Adapters.

In the MHA associated with i^{th} layer, task-specific hidden representation $h^t_{i-1} \in \mathbb{R}^{m\times d}$ from the previous layer is transformed into query Q^t_i, key K^t_i and, value V^t_i using $W^Q_i, W^K_i, W^V_i \in \mathbb{R}^{d\times d}$ transformation matrices.

$$A^t_i = h^t_{i-1} W^A_i \quad where\ A \in \{Q, K, V\} \tag{1}$$

The parameters W^Q_i, W^K_i, W^V_i are obtained from a PLM's i^{th} layer. After this, each A^t_i is divided into n_h heads, $A^t_i = [A^t_{i,1}, A^t_{i,2}, \cdots, A^t_{i,n_h}]$ where $A^t_{i,j} \in \mathbb{R}^{m\times d_h}$ and $d_h = d/n_h$. Now, we perform task-specific variations in the i^{th} layer MHA by concatenating the task-specific prefixes with these keys and values. Task-specific prefixes can be learnable parameters or generated according to the MLP re-parameterization. Here, a lower-dimensional prefix ($\hat{p}^t_i$) is passed through an MLP, and then the resulting prefixes $P^t_i = MLP(\hat{p}^t_i; \psi^t_i)$ are added to keys and values of this layer. ψ^t_i are the parameters associated with the MLP used to generate prefixes. The task-specific prefixes $P^t_i = \{P^{K,t}_i, P^{V,t}_i\} \in \mathbb{R}^{n\times d}$ are concatenated with K^t_i and V^t_i. To perform head-wise concatenation, task-specific prefixes are divided into h heads $P^{B,t}_i = [P^{B,t}_{i1}, P^{B,t}_{i,2}, \cdots, P^{B,t}_{i,h}]$ where $B \in \{K, V\}$, and $P^{B,t}_{i,j} \in \mathbb{R}^{n\times d_h}\ \forall\ j = \{1, 2, \cdots, h\}$. The new keys and values for i^{th} layer and t^{th} task are

$$\bar{B}^t_i = \Big[[P^{B,t}_{i,1}, B^t_{i,1}], [P^{B,t}_{i,2}, B^t_{i,2}], \cdots, [P^{B,t}_{i,h}, B^t_{i,h}]\Big] \tag{2}$$

where $B \in \{K, V\}$ and $\bar{B}^t_i \in \mathbb{R}^{(n+m)\times d}$. After the Prefix concatenation, it is passed to the attention block. Consider $\bar{h}^t_i$ is the output of the MHA block of i^{th} layer.

$$\bar{h}^t_i = \mathit{MHA}(Q_i, \bar{K}^t_i, \bar{V}^t_i) = concat(head^t_{i1}, head^t_{i2}, \cdots, head^t_{ih}) W^O_i + h^t_{(i-1)} \tag{3}$$

Here, $concat(\cdots)$ is a function that symbolises concatenation operation. $W^O_i \in \mathbb{R}^{d\times d}$ is a projection matrix applied after concatenating the heads and is obtained from the PLM. The heads are obtained as

$$\begin{aligned} head^t_{ij} &= Attn(Q_{ij}, \bar{K}^t_{ij}, \bar{V}^t_{ij}) \\ Attn(Q, \bar{K}, \bar{V}) &= Softmax(Q\bar{K}^T/\sqrt{d_h})\bar{V} \end{aligned} \tag{4}$$

The hidden representation obtained after the MHA $\bar{h}_i^t$ is passed through the backbone transformer's FFN layer to obtain $\hat{h}_i^t = FFN(\bar{h}_i^t)$. After FFN, $\hat{h}_i^t$ is passed through a task-specific adapter to capture task-specific knowledge and the consequent representation is obtained as $\bar{\bar{h}}_i^t = \mathcal{A}_i^t(\hat{h}_i^t) = \hat{h}_i^t + {U_i^t}^T \sigma({S_i^t}^T \hat{h}_i^t)$. Let Φ_i^t denote the collection of adapter parameters $U_i^t \in \mathbb{R}^{r \times d}$ and $S_i^t \in \mathbb{R}^{r \times d}$ associated with t^{th} task and i^{th} layer, where r represents the reduced dimension associated with the adapter. The final output of the i^{th} layer of the transformer encoder block h_i^t is given as $h_i^t = \bar{h}_i^t + \bar{\bar{h}}_i^t$. The output from the final layer of the transformer block is passed through the task-specific classification head to get the probability distribution over the classes p_k^t.

B.2. Architecture Details

Our approaches, AdaPrefix and AdaPrefix++, are based on an adapter, prefixes and hypernetwork. Each of these components has different parameters that need to be chosen. In the sections below, we provide details about the choices we have considered for our experiments.

B.2.1 AdaPrefix

AdaPrefix architecture is a combination of adapters and prefixes. For adapters, we have considered the down-scaling dimension (denoted by *reduction_factor*) as an embedding dimension of the PLM divided by 8, i.e., reduction_factor will be 8 (Appendix E). We used ReLU non-linearity in adapters and initialized adapter weights using *mam_initilization* as per [3]. Each adapter accounts for approximately 2% of the backbone PLM parameters per task. In the case of prefixes, the primary decision to be considered is the prefix_length, i.e., the number of prefix vectors to be concatenated with keys and values of each MHA layer. We fixed the prefix_length to 15 (Appendix E). Additionally, we used a two-layer MLP with ReLU non-linearity to perform reparameterization for the prefixes. Each set of prefixes takes around 0.4% of the backbone PLM parameters. AdaPrefix takes around 2.4% parameters of the backbone PLM per task.

B.2.2 AdaPrefix++

combines adapters, prefixes and hypernetwork. The choices for adapters and prefixes remain consistent with AdaPrefix. For the hypernetwork, we employed a two-layered MLP with ReLU non-linearity. We determined the layer-embedding and task-embedding dimensions to be 64, chosen from the set $32, 64, 128$, resulting in an input dimension of 128 for the hypernetwork. The output dimension of the hypernetwork is equal to the embedding dimension of the PLM. Adaprefix++ takes approximately 2.07% task-specific parameters of the backbone and 0.63% common parameters across all tasks. In total, the number of parameters in the case of AdaPrefix++ is $(2.07T + 0.63)\%$, where T is the total number of tasks.

B.2.3 Other Baselines

We had a different classifier for Sequential Finetuning with backbone frozen (FT-seq-frozen), and we only trained the classifier head on the downstream task. For CIL in FT-seq-frozen, we used the same entropy-based choice that was used in *AdaPrefix* and *AdaPrefix++*. For EWC, we set the regularization constant to 10 in our experiments. Regarding ER, we considered 100 samples per class during our experiments. For Adapters, we trained the adapters and classifiers for each task, and for CIL inference, we used entropy-based scores to choose the task-id. As for LAE-Prefix and LAE-Adapter, we used the same approach as provided in [2]. For TIL, we trained different classifiers for different tasks. As for L2P, DualPrompt, CODA-Prompt, S-Prompt and HiDe-Prompt, we considered the setup provided in the paper [8]. All these prompt-based approaches were trained in a CIL fashion. We introduced distinct classifiers for different tasks to ensure a fair comparison in the TIL scenario. Depending on the provided task IDs, we selected the appropriate classifier during TIL inference. CIL inference was done according to the algorithms provided by their respective papers [7–11].

B.3. Optimization Details

We used Adam Optimizer for both approaches (with β_1 & β_2 values equal to 0.9 & 0.999 respectively) with cosine decay of the learning rate. The learning rate was determined as $0.001 \times (batch\_size/256)$ across all setups, consistent with recent literature [8–11]. Image inputs were resized to 224×224 and normalized to $[0, 1]$ for all the experiments. We considered the batch size of 128 for all our experiments, except for ViT-L, which had a batch size of 64 because of limited memory. For training of our model, we used a single GPU of NVIDIA V100, and it took 42 mins to train each of AdaPrefix and AdaPrefix++, with ViT-B backbone on Split-CIFAR100 setup for five epochs. We chose the appropriate number of epochs for different backbone sizes to achieve convergence. For ViT-L, in all datasets, we considered three epochs; for ViT-B, in all cases, we considered five epochs; for DeiT-S, ten epochs; and DeiT-T, 15 epochs.

For all the experiments, image inputs were resized to 224×224 and normalized to $[0, 1]$ We considered the batch size of 128 for all our experiments, except for ViT-L, where we had a batch size of 64, because of limited memory. For training of our model, we used a single GPU of NVIDIA V100, and it took 44 mins to train each, AdaPrefix and AdaPrefix++, with ViT-B backbone on Split-CIFAR100 setup for five epochs. We chose the appropriate number of epochs for different backbone sizes to achieve convergence. For ViT-L, in all datasets, we considered three epochs; for ViT-B, in all cases, we considered five epochs; for DeiT-S, ten epochs; and DeiT-T, 15 epochs.

C. Different Layer Embeddings

Layer embeddings are the identifiers provided to the hypernetwork to give it an understanding of the hierarchical behaviour of different layers of the main network, in our case, the PLM. These components are independent of tasks and require to be task-agnostic and common across all the tasks. Good embedding will lead to better layer segregation and provide almost all the information about the hierarchy, which helps the hypernetwork to generate properly.

To understand this effect, we tried 5 different kinds of layer embeddings. Two among them are learnable, and 3 are fixed. 1) **full learnable**: We initialize random layer embeddings and let them get trained along with all other parameters for all the tasks. This might have an adverse effect of catastrophic forgetting. 2) **first learnable**: We initialize random layer embeddings and let them get trained with all other parameters only for the first task. The remaining tasks are kept frozen. This reduces the catastrophic forgetting in layer embeddings, but the layer embeddings will have a certain knowledge of the first task, which might adversely affect further tasks. 3) **fixed random**: We initialize random layer embeddings and then freeze it for all the tasks. This makes it task-agnostic and fixed for all tasks. However, it fails to provide any extra information about the hierarchy of the main architecture. 4) **fixed sine**: We initialize sinusoidal embeddings with each layer embeddings having a value associated with different sine wave frequencies. Specifically, as we go deeper into the main network layers, the higher sine wave frequencies are. After initializing, they are kept fixed for all tasks. 5) **fixed One-Hot**: We initialize each layer embedding with a one-hot encoding of the corresponding layer number. For example, layer embeddings for 3^{rd} layer will be $[0, 0, 1, 0, 0, \cdots, 0]^T$. If the layer embedding has a dimension, $n \times n_L$, the corresponding column will be $[1, 1, \cdots, 1]^T$, and all other values will be zeroed. After initializing these layer embeddings are kept fixed for the whole training period.

Tab. 2 provides the performance of our approach *AdaPrefix++* on different backbones and datasets for all the different initialization of embeddings. We can observe from Tab. 2 that **fixed One-Hot** performs on par or better in all datasets for TIL and CIL scenarios. To get uniformity in all the experiments, we fixed the layer embeddings to fixed One-Hot for all the other experiments.

Table 2. This table provides average accuracy (↑) of *AdaPrefix++* for different backbone PLM, on different datasets, for different initialization of layer embeddings. Details related to all the layer embeddings are provided in Appendix C.

Layer Embeddings	**TIL**			**CIL**		
	ViT-B	**DeiT-S**	**DeiT-T**	**ViT-B**	**DeiT-S**	**DeiT-T**
			CIFAR100			
full learnable	97.04	96.04	92.89	96.28	**95.37**	91.25
first learnable	97.71	95.69	92.90	96.46	95.17	91.48
fixed random	97.36	**95.94**	93.06	96.53	95.18	91.67
fixed sine	97.64	95.58	92.31	96.39	94.77	90.69
fixed One-Hot	**97.76**	95.88	**93.24**	**96.81**	94.98	**91.90**
			ImageNet-R			
full learnable	89.50	85.31	74.11	83.78	80.48	66.86
first learnable	88.84	85.27	74.02	82.73	**80.59**	66.78
fixed random	89.09	**85.46**	74.16	83.51	80.64	66.78
fixed sine	89.22	85.09	72.93	**84.41**	79.55	66.04
fixed One-Hot	**89.51**	85.25	**74.24**	84.26	79.81	**67.80**

D. Different Layer Importance

In this section, we experiment with our approach *AdaPrefix++* for different layers of a PLM architecture. Adding AdaPrefix++ to different layers performs differently. It is also important to understand whether is it necessary to add AdaPrefix++ to all the layers and also to what extent we should add it to get sufficient performance. In Tab. 3, we showed performance on multiple combinations of layers (not all possible) to get a better insight into this matter. We tried different combinations like final 1 layer (f1), initial 1 layer (i1), final 4 layers (f4), initial 4 layers (i4), final 3 and initial 3 layers (i3f3), all odd layers (odd), all even layers (even) and finally all layers (all).

We infer from Tab. 3 that we need to add to the final layers for better CIL performance. Adding to initial layers improves TIL performance, but the TIL-to-CIL performance drop is huge, as visible for i1 and i4 in the table. We can also infer that for smaller models, adding it to more layers has also helped in increasing the performance. In the case of ImageNet-R, it's visible that adding it to all the layers performs best for all scenarios. Whereas in the case of CIFAR100, for ViT-B backbone, alternate layers also perform better in TIL, whereas on Par in CIL. To have uniformity in all the experiments, we added *AdaPrefix++* to all the layers of the PLM, as it provides better performance in almost all the cases.

Table 3. This table provides average accuracy (↑) of *AdaPrefix++* when added to different layers. f# = Final # Layers, i# = Initial # Layers

Different Layers	**TIL**			**CIL**		
	ViT-B	**DeiT-S**	**DeiT-T**	**ViT-B**	**DeiT-S**	**DeiT-T**
			CIFAR100			
f1	94.74	85.55	70.84	88.22	74.58	52.62
i1	97.75	94.89	78.98	28.09	29.96	20.71
f4	97.01	91.01	84.37	95.99	88.23	76.04
i4	**98.05**	95.66	88.77	57.04	54.97	39.25
i3f3	97.81	94.85	89.23	96.30	92.48	80.63
odd	**98.05**	95.72	88.71	96.41	94.48	82.09
even	97.91	**95.74**	88.20	95.05	93.39	79.01
all	97.71	95.69	**92.90**	**96.46**	**95.17**	**91.48**
			ImageNet-R			
f1	79.66	67.21	47.23	64.97	41.77	30.30
i1	87.85	80.60	68.85	21.96	20.61	16.86
f4	84.87	76.66	56.34	78.87	67.61	46.27
i4	88.57	82.96	68.84	37.80	37.47	27.12
i3f3	87.19	82.35	65.51	80.82	72.27	48.17
odd	89.11	83.96	67.53	81.24	74.27	55.97
even	88.76	84.75	68.65	74.31	73.61	50.31
all	**88.84**	**85.27**	**74.02**	**82.73**	**80.59**	**66.78**

E. Combination of Adapter and Prefixes Hyperparameters

AdaPrefix and AdaPrefix++ are formed by a combination of adapters and prefixes. Both adapters and prefixes have their own hyperparameters, which must be tuned for these combinations. The hyperparameter associated with the adapter is its reduction factor (Main Paper Sec 3.3). The hyperparameter associated with prefixes is its prefix length (Main Paper Sec 3.2). To get optimal hyperparameter values for our setup, we performed a grid search on all possible pairs formed using adapter reduction factor set $\{2, 4, 8, 16, 32\}$ and prefix length set $\{10, 15, 20, 30, 40, 50, 60, 80, 100\}$.

Fig. 1 shows a heatmap showing the performance of AdaPrefix for different combinations of hyperparameters on CIFAR100 datasets with a backbone of ViT-B. From the heatmap, we can observe that we got optimal performance with an adapter reduction factor of 8 and a prefix length of 15. We fixed the adapter reduction factor to 8 for all the experiments and the prefix length to 15.

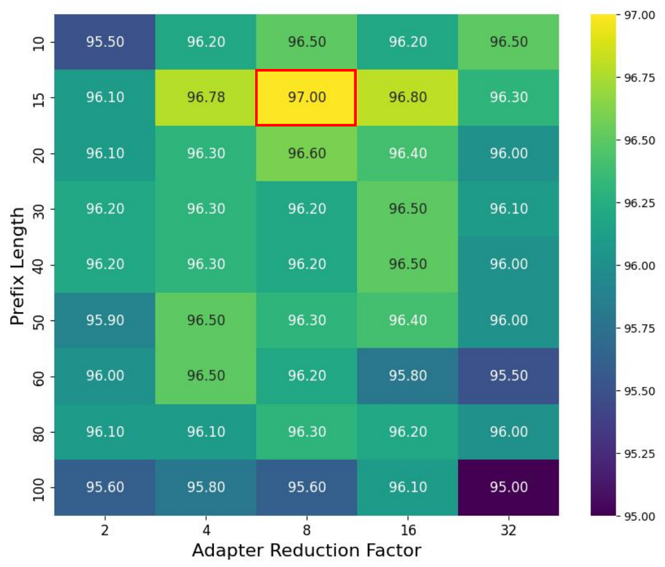

Figure 1. This heatmap shows a search over the hyperparameters of AdaPrefix [*reduction factor* of adapters (X-axis) and a *prefix length* of prefixes (Y-axis)]. We performed a grid search over the hyperparameters to determine which combination works the best (according to average accuracy). This plot considers ViT-B as the backbone and Split-CIFAR100 as the dataset to perform the experiment (TIL setting). In this case, the choice of prefix length as 15 and the reduction factor as 8 gave the best results (highlighted with red box).

F. Results

This section provides results related to different benchmark datasets, Split-CIFAR100, Split-ImageNet-R, 5-Datasets and CDDB-Hard. We give their average accuracy and variances computed for three random initializations. We also provide results on the performance change for different orders of tasks for AdaPrefix++. No change occurs for AdaPrefix as it is based on the parameter-isolation method. For CIFAR100, ImageNet-R, 5-datasets and CDDB-Hard, we randomly shuffled the task orders into five permutations.

F.1. Computational Overheads

This section details parameter growth (in percentage) for different tasks. Tab. 4 shows that AdaPrefix++ reduces parameter growth compared to AdaPrefix with a boost in performance because of knowledge transfer across tasks. The time taken to train each task for AdaPrefix++ is also less compared to Adapters, Prefix and AdaPrefix.

Table 4. This table provides values for CIFAR100 dataset, with VIT-B as backbone. The per cent of parameter growth is provided compared to backbone architecture. The formula used is [(Method Parameters) / (Backbone Parameter)] * 100. Running time is provided for training on a single GPU, NVIDIA Tesla V100, with a batch size of 128.

	Adapter	Prefix	Hnet+Prefix	**AdaPrefix**	**AdaPrefix++**
Task-Specific Parameter per task (in percent)	2.15	0.9	0.003	2.4	2.07
Common Parameters across all task (in percent)	-	-	0.43	-	0.43
Total Parameters after T tasks (in percent)	2.15 $\times\|T\|$	0.9 $\times\|T\|$	0.003 $\times\|T\|$ + 0.43	2.4 $\times\|T\|$	2.07 $\times\|T\|$ + 0.43
Training Time per task	51 mins	46 mins	42 mins	47 mins	44 mins
Inference Time per 128 image	3.98 ms	4 ms	3.69 ms	3.98 ms	3.69 ms
Memory Consumption (MB)	389.50	345.33	323.80	444.50	403.35

F.2. Split-CIFAR100

For the Split-CIFAR100 dataset, we consider the CIFAR100 [5] dataset (100 classes in total) and divide it into ten different disjoint tasks, each containing ten classes. In CIFAR100, small-scale examples of 32×32 pixels are present. Tab. 5 presents results for average accuracy over all tasks for different baselines and our approaches. Tab. 6 provides results for five different task orders for Split-CIFAR100 dataset.

Table 5. Average Accuracy (↑) of all the baselines along with our approaches, on CIFAR100 dataset is provided in this table.

Methods	**ViT-L**	**ViT-B**	**DeiT-S**	**DeiT-T**
		Task Incremental Learning (TIL)		
FT-seq-frozen	83.52 ± 0.31	80.25 ± 0.41	62.48 ± 0.41	55.66 ± 0.87
EWC	85.25 ± 0.52	81.49 ± 0.81	62.94 ± 0.57	56.68 ± 0.75
ER	89.21 ± 0.21	85.23 ± 0.85	69.08 ± 0.25	59.45 ± 0.81
Adapter	97.64 ± 0.12	96.45 ± 0.11	93.20 ± 0.17	92.00 ± 0.13
LAE-Prefix	97.01 ± 0.45	96.89 ± 0.31	91.15 ± 0.46	89.69 ± 0.74
LAE-Adapter	97.51 ± 0.37	96.89 ± 0.29	92.44 ± 0.64	89.24 ± 0.71
L2P	94.95 ± 0.15	94.35 ± 0.19	88.33 ± 0.22	84.02 ± 0.11
DualPrompt	95.12 ± 0.11	94.90 ± 0.21	90.14 ± 0.21	85.16 ± 0.21
CODA-Prompt	96.00 ± 0.53	95.52 ± 0.76	89.86 ± 0.85	85.35 ± 0.69
S-Prompt	96.99 ± 0.18	96.85 ± 0.13	92.45 ± 0.13	85.90 ± 0.21
Hide-Prompt	**98.14 ±0.28**	97.50 ± 0.19	95.25 ± 0.16	87.00 ± 0.17
AdaPrefix	97.82 ± 0.12	97.10 ± 0.31	94.50 ± 0.35	92.92 ± 0.26
AdaPrefix++	**98.14** ± **0.14**	**97.76** ± **0.13**	**95.88** ± **0.19**	**93.24** ± **0.22**
		Class Incremental Learning (CIL)		
FT-seq-frozen	17.94 ± 0.98	17.59 ± 0.76	15.28 ± 0.89	12.85 ± 0.78
EWC	64.12 ± 0.64	59.49 ± 0.21	62.94 ± 0.48	56.68 ± 0.72
ER	76.25 ± 0.44	71.53 ± 0.14	69.08 ± 0.67	59.45 ± 0.13
Adapter	96.79± 0.21	94.99 ± 0.23	92.11± 0.29	90.10 ± 0.31
LAE-Prefix	89.75 ± 0.35	89.25 ± 0.48	86.15 ± 0.41	84.69 ± 0.44
LAE-Adapter	90.01 ± 0.52	89.59 ± 0.57	86.45 ± 0.51	80.51 ± 0.62
L2P	84.20 ± 0.17	83.06 ± 0.32	78.21 ± 0.31	71.02 ± 0.21
DualPrompt	87.10 ± 0.18	86.60 ± 0.15	81.14 ± 0.10	72.46 ± 0.25
CODA-Prompt	88.56 ± 0.63	86.94 ± 0.75	81.86 ± 0.14	70.35 ± 0.19
S-Prompt	89.41 ± 0.11	88.81 ± 0.14	84.45 ± 0.86	79.90 ± 0.41
Hide-Prompt	93.25 ± 0.28	92.61 ± 0.14	86.25 ± 0.42	80.11 ± 0.10
AdaPrefix	96.92 ± 0.22	96.11 ± 0.19	93.20 ± 0.12	89.75 ± 0.22
AdaPrefix++	**96.99** ± **0.05**	**96.81** ± **0.37**	**94.98** ± **0.18**	**91.90** ± **0.31**

Table 6. We provide Average Accuracy over tasks for different task orders for AdaPrefix++ on CIFAR100, with ViT-B backbone

Different Orders	Order 1	Order 2	Order 3	Order 4	Order 5	**Average**
TIL	97.71	97.03	97.33	97.35	97.55	97.39
CIL	96.75	96.30	96.68	96.70	96.62	96.61

F.3. Split-ImageNet-R

For the Split-ImageNet-R dataset, we considered the ImageNet-R [4] dataset (200 classes in total) and divided it into ten different disjoint tasks, each containing 20 classes. ImageNet-R contains 200 class large-scale images of hard examples from ImageNet [1] dataset or newly collected examples of different styles. Tab. 7 presents results for average accuracy over all tasks for different baselines and our approaches. Tab. 8 provides results for five different task orders for Split-ImageNet-R dataset. Tab. 9 provides results of Imagenet-R in a more challenging setting where different tasks have different amounts of complexity. We varied the task complexity by introducing different numbers of classes in different tasks. The table shows that AdaPrefix++ performs better in all the cases for both TIL and CIL settings compared to Hide-Prompt.

Table 7. Average Accuracy (↑) of all the baselines along with our approaches on the ImageNet-R dataset is provided in this table.

Methods	**ViT-L**	**ViT-B**	**DeiT-S**	**DeiT-T**
		Task Incremental Learning (TIL)		
FT-seq-frozen	64.58 ± 0.89	63.15 ± 0.87	54.78 ± 0.78	50.17 ± 0.94
EWC	64.25 ± 0.81	63.12 ± 0.12	51.10 ± 0.38	49.22 ± 0.15
ER	71.11 ± 0.23	71.10 ± 0.58	60.12 ± 0.56	58.10 ± 0.13
Adapter	87.87 ± 0.18	87.50 ± 0.12	84.00 ± 0.15	72.50 ± 0.22
LAE-Prefix	83.11 ± 0.28	81.89 ± 0.26	73.14 ± 0.29	70.01 ± 0.34
LAE-Adapter	88.41 ± 0.35	88.14 ± 0.31	85.10 ± 0.36	72.51 ± 0.39
L2P	76.99 ± 0.11	75.65 ± 0.11	69.25 ± 0.16	61.25 ± 0.24
DualPrompt	76.54 ± 0.17	75.99 ± 0.26	69.54 ± 0.14	63.95 ± 0.29
CODA-Prompt	82.21 ± 0.45	79.03 ± 0.85	72.40 ± 0.75	66.25 ± 0.84
S-Prompt	79.24 ± 0.21	77.68 ± 0.21	72.21 ± 0.13	66.21 ± 0.21
Hide-Prompt	87.54 ± 0.12	85.06 ± 0.31	79.21 ± 0.12	69.99 ± 0.13
AdaPrefix	88.31 ± 0.32	87.95 ± 0.12	84.45 ± 0.16	73.72 ± 0.13
AdaPrefix++	**89.79** ± **0.31**	**89.51** ± **0.19**	**85.25** ± **0.45**	**74.24** ± **0.25**
		Class Incremental Learning (CIL)		
FT-seq-frozen	15.93 ± 0.94	14.63 ± 0.89	13.98 ± 1.01	10.11 ± 1.56
EWC	52.10 ± 0.82	49.11 ± 0.78	45.28 ± 0.98	41.02 ± 0.82
ER	57.10 ± 0.64	49.55 ± 0.62	49.52 ± 0.85	45.25 ± 0.56
Adapter	81.25 ± 0.15	81.11 ± 0.18	72.22 ± 0.13	62.10 ± 0.19
LAE-Prefix	78.84 ± 0.82	77.55 ± 0.78	72.11 ± 0.74	60.32 ± 0.82
LAE-Adapter	80.11 ± 0.54	79.07 ± 0.58	74.51 ± 0.57	61.94 ± 0.49
L2P	71.99 ± 0.19	71.65 ± 0.21	62.25 ± 0.45	59.25 ± 0.12
DualPrompt	72.54 ± 0.41	71.79 ± 0.16	62.54 ± 0.32	59.95 ± 0.15
CODA-Prompt	75.21 ± 0.56	75.03 ± 0.48	65.40 ± 0.65	61.25 ± 0.52
S-Prompt	75.24 ± 0.14	74.68 ± 0.12	63.21 ± 0.31	60.21 ± 0.11
Hide-Prompt	77.25 ± 0.31	76.45 ± 0.13	72.21 ± 0.26	63.99 ± 0.14
AdaPrefix	81.95 ± 0.82	83.13 ± 0.46	79.95 ± 0.14	64.31 ± 0.35
AdaPrefix++	**84.39** ± **0.64**	**84.26** ± **0.19**	**79.81** ± **0.21**	**67.80** ± **0.31**

Table 8. We provide Average Accuracy over tasks for different task orders for AdaPrefix++ on ImageNet-R, with ViT-B backbone

Different Orders	Order 1	Order 2	Order 3	Order 4	Order 5	**Average**
TIL	89.63	89.90	89.27	89.25	89.79	89.57
CIL	84.57	84.62	83.39	83.68	83.94	84.04

Table 9. This table provides results of the ImageNet-R dataset on ViT-Base architecture with varying numbers of classes on different tasks. We provide test set accuracy for each of the tasks after being trained completely over the sequence of tasks.

	Different Tasks	10 Cls	20 Cls	30 Cls	20 Cls	10 Cls	10 Cls	AVG
					ImageNet-R			
TIL	Hide-Prompt	90.12	83.54	83.12	86.47	**87.75**	83.12	85.69
	AdaPrefix++	**90.23**	**83.94**	**85.32**	**86.71**	87.26	**84.32**	**86.30**
CIL	Hide-Prompt	80.82	71.35	71.58	79.54	80.14	80.10	77.26
	AdaPrefix++	**87.82**	**75.97**	**76.65**	**83.54**	**83.64**	**82.94**	**81.76**

F.4. 5-Datasets

The five datasets [11] include CIFAR10, SVHN, FashionMNIST, MNIST, and notMNIST datasets, each provided as an incremental task. This dataset setup provides us with a more robust understanding of the impact of different tasks on our approaches. Tab. 10 presents results for average accuracy over all tasks for different baselines and our methods. Tab. 11 provides results for five different task orders for these five datasets.

Table 10. Average Accuracy (↑) of all the baselines, along with our approaches, on 5-datasets is provided in this table.

Methods	**ViT-L**	**ViT-B**	**DeiT-S**	**DeiT-T**
		Task Incremental Learning (TIL)		
FT-seq-frozen	44.44 ± 0.85	44.12 ± 0.75	35.12 ± 0.94	29.56 ± 0.63
EWC	46.23 ± 0.25	45.21 ± 0.45	36.21 ± 0.54	30.25 ± 0.53
ER	69.12 ± 0.45	68.1 ± 0.25	62.27 ± 0.56	58.10 ± 0.29
Adapter	95.11 ± 0.32	94.81 ± 0.77	89.17 ± 0.68	84.61 ± 0.12
LAE-Prefix	89.52 ± 0.65	88.27 ± 0.79	78.71 ± 0.81	72.65 ± 0.75
LAE-Adapter	94.89 ± 0.19	93.75 ± 0.22	87.99 ± 0.09	85.41 ± 0.12
L2P	91.48 ± 0.11	90.23 ± 0.18	82.14 ± 0.12	78.56 ± 0.15
DualPrompt	91.52 ± 0.15	90.24 ± 0.16	83.11 ± 0.15	78.35 ± 0.14
CODA-Prompt	93.15 ± 0.25	92.59 ± 0.59	84.21 ± 0.69	79.28 ± 0.95
S-Prompt	93.65 ± 0.16	93.21 ± 0.14	86.24 ± 0.15	80.41 ± 0.15
Hide-Prompt	$\mathbf{95.90 \pm 0.18}$	$\mathbf{95.20 \pm 0.19}$	91.58 ± 0.11	85.01 ± 0.14
AdaPrefix	95.24 ± 0.13	94.90 ± 0.21	$\mathbf{91.58 \pm 0.25}$	85.10 ± 0.14
AdaPrefix++	95.28 ± 0.15	94.76 ± 0.32	90.52 ± 0.15	$\mathbf{85.28 \pm 0.32}$
		Class Incremental Learning (CIL)		
FT-seq-frozen	13.01 ± 0.59	12.15 ± 0.77	10.89 ± 0.49	10.01 ± 0.99
EWC	33.59 ± 0.29	33.04 ± 0.77	30.41 ± 0.54	25.61 ± 0.12
ER	50.12 ± 0.38	48.21 ± 0.56	45.23 ± 0.63	43.02 ± 0.56
Adapter	88.45 ± 0.12	88.01 ± 0.17	85.25 ± 0.19	80.99 ± 0.13
LAE-Prefix	86.95 ± 0.29	85.49 ± 0.21	76.25 ± 0.27	70.55 ± 0.22
LAE-Adapter	89.01 ± 0.29	88.17 ± 0.22	86.55 ± 0.34	81.25 ± 0.37
L2P	84.17 ± 0.11	83.06 ± 0.12	78.28 ± 0.11	76.21 ± 0.15
DualPrompt	86.90 ± 0.12	86.60 ± 0.13	79.26 ± 0.15	77.21 ± 0.15
CODA-Prompt	88.90 ± 0.37	86.94 ± 0.48	81.11 ± 0.65	79.64 ± 0.49
S-Prompt	89.11 ± 0.12	88.81 ± 0.19	85.21 ± 0.25	79.60 ± 0.13
Hide-Prompt	$\mathbf{92.91 \pm 0.16}$	$\mathbf{92.61 \pm 0.13}$	89.21 ± 0.13	81.20 ± 0.17
AdaPrefix	91.99 ± 0.19	91.94 ± 0.14	89.12 ± 0.19	$\mathbf{81.21 \pm 0.21}$
AdaPrefix++	91.87 ± 0.13	91.40 ± 0.18	$\mathbf{89.25 \pm 0.11}$	81.10 ± 0.17

Table 11. We provide Average Accuracy over all tasks for different task orders for AdaPrefix++ on 5-datasets, with ViT-B backbone

Different Orders	Order 1	Order 2	Order 3	Order 4	Order 5	**Average**
TIL	94.15	94.52	94.74	94.53	94.62	94.512
CIL	91.32	90.09	92.01	91.10	91.52	91.37

F.5. CDDB-Hard

The CDDB-Hard [6] dataset is related to continual deep fake detection. In this dataset, we are provided fake images generated by different generative models and images from different generative models are considered from different domains. CDDB-Hard contains images from Gaussian-GAN, BigGAN, WildDeepFakes, Whichface-Real and SAN. More details about these are present in [6]. Tab. 12 compares our approach AdaPrefix++ with all the other baseline approaches in a DIL scenario on the CDDB-Hard dataset. Tab. 13 provides results for five different task orders for the CDDB-Hard dataset.

Table 12. Average Accuracy Results (↑) on CDDB-Hard deep-fake detection dataset in a domain incremental setting (DIL)

Methods	**ViT-L**	**ViT-B**	**DeiT-S**	**DeiT-T**
EWC	51.10 ± 0.78	50.59 ± 0.88	40.25 ± 0.56	29.34 ± 0.98
ER	74.01 ± 0.69	73.90 ± 0.75	65.24 ± 0.46	51.24 ± 0.81
LAE-Adapter	75.11 ± 0.51	74.01 ± 0.45	69.24 ± 0.16	64.99 ± 0.31
L2P	62.10 ± 0.16	61.28 ± 0.21	56.47 ± 0.11	42.14 ± 0.24
DualPrompt	61.99 ± 0.21	61.39 ± 0.16	57.01 ± 0.17	42.11 ± 0.17
CODA-Prompt	66.14 ± 0.56	65.22 ± 0.54	59.14 ± 0.68	47.14 ± 0.78
S-Prompt	75.12 ± 0.19	74.51 ± 0.14	70.25 ± 0.19	63.10 ± 0.19
AdaPrefix++	$\mathbf{78.45 \pm 0.13}$	$\mathbf{77.06 \pm 0.17}$	$\mathbf{75.21 \pm 0.27}$	$\mathbf{67.12 \pm 0.31}$

Table 13. We provide Average Accuracy over all tasks for different task orders for AdaPrefix++ on CDDB-Hard, with ViT-B backbone

Different Orders	Order 1	Order 2	Order 3	Order 4	Order 5	**Average**
DIL	76.59	77.06	76.99	76.92	77.38	76.988

Table 14. In this table, we provide domain-wise average accuracy for S-Prompt [9] and our approach AdaPrefix++ (for TIL (with domain-ids) and DIL (without domain-ids)). We provide results on a sequence of 8 domains taken for CDDB dataset [6] with ViT-Base as the backbone.

	gaugan	biggan	deepfake	imle	crn	wild	whichfaceisreal	san	AVG
S-Prompt	97.60	94.12	91.54	100.0	98.56	76.21	73.56	52.31	85.49
AdaPrefix++ (TIL)	98.90	95.00	94.08	100.0	99.53	77.56	80.00	56.67	87.71
AdaPrefix++ (DIL)	99.30	95.125	93.99	100.0	99.49	76.20	72.50	54.44	86.38

G. Experiments on NLP Datasets (CL Benchmarks)

In this section, we provide the results of our approach AdaPrefix and AdaPrefix++ on one of the popular Continual Learning benchmarks in NLP, CL Benchmark. It contains 5 tasks: Yelp (sentiment analysis), AG News (topic classification), Amazon (sentiment analysis), DBpedia (topic classification), and Yahoo (topic classification). All of these tasks use accuracy as a metric. To reduce the computational cost, we further sampled 1000 samples per class for all datasets; since test sets are not available for all datasets, we used validation sets as our test data across all the experiments and held our 500 samples from training data as validation. All of these experiments are done on the T5-Small model. Tab. 15 provides results for all these tasks on our approach and a few baselines. From this table, we can see that our approach also works well for NLP tasks. We can also see that AdaPrefix works better than another approach, and AdaPrefix++ holds on to that accuracy even with a reduction in the number of parameters.

Table 15

	CL Benchmark					**Average**
	Yelp	Amazon	DBPedia	Yahoo	AG News	
EWC	27.71	27.52	25.38	15.97	47.97	31.00
Prefix	45.60	33.52	96.51	69.76	86.20	66.00
AdaPrefix	49.84	**41.60**	**97.06**	71.20	88.00	**70.00**
Hnet+Prefix	47.52	37.71	96.18	70.11	85.80	67.00
AdaPrefix++	**50.05**	41.09	96.89	**71.64**	**88.03**	**70.00**

References

[1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 8

[2] Qiankun Gao, Chen Zhao, Yifan Sun, Teng Xi, Gang Zhang, Bernard Ghanem, and Jian Zhang. A unified continual learning framework with general parameter-efficient tuning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11483–11493, 2023. 3

[3] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*, 2022. 3

[4] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, Dawn Song, Jacob Steinhardt, and Justin Gilmer. The many faces of robustness: A critical analysis of out-of-distribution generalization. *ICCV*, 2021. 8

[5] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. 7

[6] Chuqiao Li, Zhiwu Huang, Danda Pani Paudel, Yabin Wang, Mohamad Shahbazi, Xiaopeng Hong, and Luc Van Gool. A continual deepfake detection benchmark: Dataset, methods, and essentials. In *Winter Conference on Applications of Computer Vision (WACV)*, 2023. 10

[7] James Seale Smith, Leonid Karlinsky, Vyshnavi Gutta, Paola Cascante-Bonilla, Donghyun Kim, Assaf Arbelle, Rameswar Panda, Rogerio Feris, and Zsolt Kira. Coda-prompt: Continual decomposed attention-based prompting for rehearsal-free continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11909–11919, 2023. 3

[8] Liyuan Wang, Jingyi Xie, Xingxing Zhang, Mingyi Huang, Hang Su, and Jun Zhu. Hierarchical decomposition of prompt-based continual learning: Rethinking obscured sub-optimality. *Advances in Neural Information Processing Systems*, 36, 2024. 3

[9] Yabin Wang, Zhiwu Huang, and Xiaopeng Hong. S-prompts learning with pre-trained transformers: An occam's razor for domain incremental learning. *Advances in Neural Information Processing Systems*, 35:5682–5695, 2022. 3, 10

[10] Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, et al. Dualprompt: Complementary prompting for rehearsal-free continual learning. In *European Conference on Computer Vision*, pages 631–648. Springer, 2022. 3

[11] Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 139–149, 2022. 3, 9