Supplementary Material

A. Source Code

We plan to release the source code for this work in our arXiv version[4].

B. Details of APT Weight (APT-W)

In this section we describe the details of the APT Weight (APT-W) scheme. Let $\mathcal{D} = \{D_1, \ldots, D_n\}$ be a collection of sources. Consistent with APT for each source D_i we train a prompt $p^{(i)}$ and a classifier head head_i using only the data in D_i. Then, differing with classical APT, for each source D_i we perform K-means ($K = 20$) in the embedding space to construct a set of prototypes $\mu_1^{(i)}, \ldots, \mu_K^{(i)}$. More concretely for each $(x, y) \in D_i$ we forward the input x through the transformer to get the final embedding sequence $[\mathbf{z}_L(x)]$. We use the class token embeddings $\mathbf{z}_L^{(0)}(x)$ as the vectors to perform the K-means algorithm on. Specifically we perform K-means on the set

$$\{\mathbf{z}_L^{(0)}(x) : (x, y) \in D_i\}$$

to construct the prototypes $\mu_1^{(i)}, \ldots, \mu_K^{(i)}$. At inference time, the basic forward pass for APT Weight is the same as APT. Given an instance x and a set $I = \{i_1, \ldots, i_{|I|}\} \subset [n]$ we let $\mathbf{p}^{(I)} = [p^{(i_1)}, \ldots, p^{(i_{|I|})}]$ be the concatenation of all prompt tokens corresponding to each data source in $\mathcal{D}_I$. The final output of the transformer is given by

$$[\mathbf{z}_L, \mathbf{p}_L^{(I)}] := F_\theta^L \circ \ldots \circ F_\theta^1([\mathbf{z}_0, \mathbf{p}^{(I)}])$$

where the structured attention is applied as usual. Each output token $p_L^{(i)}$ corresponding to a prompt $p^{(i)}$ is used to generate logits

$$\hat{y}^{(i)} := \mathrm{head}_i(p_L^{(i)}).$$

In contrast to APT, APT Weight will apply a weighting to the logits $\hat{y}^{(i)}$ based on the distance of the embedding of the instance x to the prototypes $\mu_1^{(i)}, \ldots, \mu_K^{(i)}$. Specifically, for each index in $i \in I$ we compute

$$d_i = \min_{k \in [K]} \|\mathbf{z}_L^{(0)} - \mu_k^{(i)}\|_2.$$

Let us denote $\mathbf{d} = (d_{i_1}, \ldots, d_{i_{|I|}})$. Then we construct a weight vector

$$w := \mathrm{softmax}(-\beta \mathbf{d})$$

where β is the inverse temperature which in our experiments we set to $\beta = 0.1$. We then form the weighted logits

$$[w_{i_1} \cdot \hat{y}^{(i_1)}, w_{i_2} \cdot \hat{y}^{(i_2)}, \ldots, w_{i_{|I|}} \cdot \hat{y}^{(i_{|I|})}].$$

[4] https://arxiv.org/abs/2302.07994

For class incremental learning problems, these weighted logits are the final logits used for prediction. For domain incremental learning problems, the logits are average pooled to form the final logits

$$\hat{y} = \frac{1}{|I|} \sum_{i \in I} w_i \cdot \hat{y}^{(i)}.$$

C. Hyperparameters

Consistent with [38] for the continual learning experiments on Split CIFAR-100 and CORe50 we train for 5 epochs. All methods in Table 3 are trained for 150 epochs. For all other experiments the paragon method (trained on the entire dataset) is trained for 150 epochs whereas the prompts in the APT method are trained for 80 epochs on their respective sources. We emphasize that the paragon is never trained for fewer epochs than the APT method to remain a true "upper bound". For the paragon prompt tuning numbers we do not use structured attention. Consistent with [35] we use 5 memory tokens at each layer for deep prompting. For the prompt tuning of the APT method structured attention is applied during both train and inference time. In all cases we optimize using the AdamW algorithm [27] with the weight decay parameter set to 0.02. We use linear warmup cosine annealing with start learning rate $1\mathrm{e}{-5}$, minimum learning rate $1\mathrm{e}{-6}$, and one warmup epoch. The base learning rates for finetuning, head-only finetuning, and prompt tuning are $1\mathrm{e}{-5}$, $5\mathrm{e}{-1}$, and $1\mathrm{e}{-1}$ respectively. We did not do any hyperparameter sweep over learning rates. The one exception is for the "Bias+Head" column in Table 3 we did a sweep over learning rates to arrive at the learning rate $5\mathrm{e}{-3}$. However, we note that this column is merely for comparison and does not concern our specific method. We use a batch size of 8 and follow the convention presented in [12] of rescaling the learning rate by the effective batch size (batch size x devices x nodes) divided by 256.

We perform data augmentation following standard practice in training ViTs and include RandAugment [3] with N=2 and M=10. However we note that we did not use Mixup [39] which is known to be a reliable way of increasing performance.

D. Dataset Details

In Table 5 we report detailed statistics for the datasets used as well as links to download the datasets.

E. Additional Ablations

Average ensembling vs. majority vote. In our APT method we chose to aggregate the individual predictions of the prompts by average ensembling. Another common ensembling method is to perform majority vote. Consistent

Table 5. **Dataset sample/class counts.** We list the number of training images, test images, and classes for each of the datasets. We also provide a link to download the data.

Dataset	**Training Images**	**Testing Images**	**# Classes**	**URL**
MIT-67 [33]	5360	1340	67	`https://web.mit.edu/torralba/www/indoor.html`
CUB-200 [36]	5994	5794	200	`https://www.vision.caltech.edu/datasets/cub_200_2011/`
Caltech-256 [13]	15418	15189	257	`https://authors.library.caltech.edu/7694/`
Oxford Pets [31]	3680	3669	37	`https://www.robots.ox.ac.uk/~vgg/data/pets/`
FGVC-Aircrafts [28]	6667	3333	100	`https://www.robots.ox.ac.uk/~vgg/data/fgvc-aircraft/`
Oxford Flowers [30]	2040	6149	102	`https://www.robots.ox.ac.uk/~vgg/data/flowers/102/`
Stanford Cars [16]	8144	8041	196	`https://ai.stanford.edu/~jkrause/cars/car_dataset.html`
CIFAR-100 [17]	50,000	10,000	100	`https://www.cs.toronto.edu/~kriz/cifar.html`
CORe50 [25, 26]	119,894	44,972	50	`https://vlomonaco.github.io/core50/`

Dataset	2 Shards	3 Shards	5 Shards	10 Shards	15 Shards	20 Shards	50 Shards
MIT-67	3.1%	0.9%	0.6%	0.7%	0.6%	0.9%	0.5%
Cub-200	3.3%	1.1%	1.5%	0.8%	1.0%	0.8%	1.3%
Caltech-256	3.0%	1.4%	0.8%	0.2%	0.6%	0.4%	0.2%
Pets	1.4%	0.2%	0.3%	0.0%	-0.1%	0.2%	0.3%
Aircrafts	6.2%	5.2%	5.0%	3.8%	3.2%	3.7%	3.0%
Flowers	0.7%	4.2%	0.2%	0.6%	0.7%	1.3%	2.8%
Stanford Cars	9.0%	7.6%	5.6%	5.5%	4.9%	5.1%	4.7%
Average	3.81%	2.94%	2.0%	1.66%	1.56%	1.77%	1.83%

Table 6. **Average vs. majority vote.** We report the accuracy of average ensembling minus the accuracy of majority vote. We observe that average ensembling uniformly outperforms majority vote.

with [1] we find that average ensembling outperforms majority vote. In Table 6 we report the performance gap of average ensembling over majority vote for the sharding experiment. We see that excluding one exceptional case, average ensembling uniformly outperforms majority vote for all datasets and all numbers of shards. The performance gain on average is in the range 1.5-3.8%.

Pretraining. To investigate how APT performs when the backbone transformer has a different pretraining, instead of using ImageNet21k we experiment with loading the VIT-B/16 from the visual encoder of the multimodal model ALBEF [20]. In Table 7 we report the accuracies of APT applied to this checkpoint. We see that the performance of APT for the ALBEF visual encoder decays more quickly as the number of shards increases relative to the ImageNet21k numbers reported in Table 2. For example for the visual encoder of ALBEF, for 10 shards only the datasets MIT-67, Caltech-256, and Pets are within 5% performance of the paragon, whereas by contrast for the ImageNet21k checkpoint all datasets except for Aircrafts and Stanford Cars are within 5% performance of paragon even when the number of shards is twice as large, namely 20. Thus we conclude that the pretraining of the backbone transformer is highly pertinent for the performance of APT. This is sensible as due to the structured attention the APT prompts do not modify the internal representations of the backbone, and thus are unable to provide compensation whenever the backbone representations are deficient.

Finetuning. While inference and storage for the APT method is less costly than ensembling finetuned models, it is worthwhile to ask how the two compare in terms of classification accuracy. In Table 8 we report the accuracies for the sharding experiment using finetuning instead of APT. Specifically we finetune separate models on each shard which are then ensembled at inference time. By comparing the results in Table 2 to the results in Table 8, we see that APT uniformly outperforms finetuning in terms of classification accuracy, and the gap becomes most pronounced as the number of shards increases. Specifically, for 20 and 50 shards APT has average accuracy of 77.3% and 73.9% respectively compared to 41.5% and 25.6% for finetuning. We believe this is due to finetuning being more susceptible to overfitting when there are fewer data in contrast to APT which uses a fixed backbone and thus has a stronger inductive bias.

Dataset	No Sharding	2 Shards	3 Shards	5 Shards	10 Shards	15 Shards	20 Shards	50 Shards
MIT-67	89.1%	87.5%	88.4%	88.9%	89.0%	88.4%	88.4%	87.3%
Cub-200	78.8%	71.6%	66.9%	57.1%	54.9%	48.5%	46.2%	39.6%
Caltech-256	91.4%	88.8%	89.2%	88.7%	88.9%	88.6%	87.8%	86.2%
Pets	91.1%	89.9%	88.2%	87.0%	86.3%	83.1%	81.0%	66.1%
Aircrafts	72.6%	60.5%	54.4%	51.1%	40.2%	38.7%	35.9%	30.7%
Flowers	93.4%	85.1%	83.1%	81.7%	80.7%	78.1%	76.2%	67.8%
Stanford Cars	83.3%	78.2%	76.5%	70.7%	63.7%	59.5%	55.9%	43.6%
Average	85.67%	80.23%	78.1%	75.03%	71.96%	69.27%	67.34%	60.19%

Table 7. **Sharding from ALBEF pretraining.** We report the accuracies for the sharding experiment using the ALBEF checkpoint.

Dataset	No Sharding	2 Shards	3 Shards	5 Shards	10 Shards	15 Shards	20 Shards	50 Shards
MIT-67	87.1%	86.1%	83.8%	81.9%	74.4%	69.9%	68.8%	44.9%
Cub-200	88.4%	81.8%	76.4%	70.9%	54.4%	42.5%	32.5%	5.9%
Caltech-256	93.5%	90.3%	87.8%	85.8%	81.0%	78.2%	74.1%	52.0%
Pets	94.5%	93.6%	92.6%	91.2%	89.7%	84.2%	81.6%	55.8%
Aircrafts	75.6%	51.1%	44.5%	36.2%	24.1%	23.0%	19.2%	12.3%
Flowers	97.4%	75.3%	56.1%	39.8%	15.6%	11.1%	2.2%	2.0%
Stanford Cars	84.3%	53.3%	39.4%	28.2%	19.2%	16.2%	11.8%	6.4%
Average	88.69%	75.93%	68.66%	62.00%	51.20%	46.44%	41.46%	25.61%

Table 8. **Sharding using finetuning.** We report the accuracy for the sharding experiment when using finetuning.