PRO-VPT: Distribution-Adaptive Visual Prompt Tuning via Prompt Relocation

Supplementary Material

This appendix presents further details and results that could not be included in the main paper due to space constraints. The content is organized as follows:

- § A provides detailed explanations of the related technical concepts.
- § B provides the detailed pseudo-code of PRO-VPT.
- § C presents the complete results of distribution adjustments and offers an in-depth analysis of the underlying nature of ADO, which motivates the nested optimization formulation.
- § D presents attempts at adding-based adjustments, which demonstrate significant instability.
- § E explains why our proposed PR strategy relocates only a single prompt for each iteration.
- § F details the derivation of the Taylor expansion for the idleness score.
- § G provides more details of implementation.
- § H presents complete results for VTAB-1k and FGVC as well as additional experimental results.
- § I presents additional visualizations and analyses.
- § J discusses the limitations of PRO-VPT and points out the potential direction for our future work.

A. Detailed Explanations of Technical Concepts

To help readers better understand the relevant technical concepts mentioned in this paper, we provide a detailed explanation of the related approaches as follows:

Vision Transformer. Given an input image $\mathbf{x}$, ViT [7, 26, 64] first divides it into n_e fixed-sized patches. Each patch is then embedded into d-dimensional latent space and combined with position encoding. The resulting set of patch embeddings is denoted as $\mathbf{E}_0 = \{\mathbf{e}_{0j} \in \mathbb{R}^d\}_{j=1}^{n_e}$. A learnable classification token $\mathbf{x}_0$ is then concatenated with these embeddings, forming the input sequence $[\mathbf{x}_0, \mathbf{E}_0]$. This sequence is then fed into a series of L Transformer blocks $\{B_i(\cdot)\}_{i=1}^{L}$ as follows:

$$[\mathbf{x}_i, \mathbf{E}_i] = B_i\big([\mathbf{x}_{i-1}, \mathbf{E}_{i-1}]\big),\ i = 1, 2, \ldots, L. \tag{i}$$

Visual Prompt Tuning. Given a set of trainable prompt tokens $\mathbf{P}$ and the prompted backbone $f_{\mathbf{P}}(\cdot)$, the overall objective of VPT [22] is to optimize these prompts for effectively adapting the PVM to downstream tasks:

$$\min_{\mathbf{P}}\ \mathbb{E}_{(\mathbf{x},y)\in\mathcal{T}_{tr}}\big[\mathcal{L}(f_{\mathbf{P}}(\mathbf{x}), y)\big]. \tag{ii}$$

This formulation specifically corresponds to Eq. (2).

Depending on how the prompt set $\mathbf{P}$ is distributed across the Transformer blocks, the standard VPT [22] can be categorized into two variants, VPT-Shallow and VPT-Deep:

VPT-Shallow. The entire set of p prompts, $\mathbf{P} = \{\mathbf{p}_k \in \mathbb{R}^d\}_{k=1}^{p}$, is introduced merely in the first block. The shallow-prompted model is formulated as:

$$[\mathbf{x}_1, \mathbf{Z}_1, \mathbf{E}_1] = B_1\big([\mathbf{x}_0, \mathbf{P}, \mathbf{E}_0]\big), \tag{iii}$$

$$[\mathbf{x}_i, \mathbf{Z}_i, \mathbf{E}_i] = B_i\big([\mathbf{x}_{i-1}, \mathbf{Z}_{i-1}, \mathbf{E}_{i-1}]\big),\ i = 2, 3, \ldots, L. \tag{iv}$$

VPT-Deep. The prompt set $\mathbf{P}$ is uniformly distributed across all blocks, where each block i is allocated a subset of m prompts, $\mathbf{P}_i = \{\mathbf{p}_{ik} \in \mathbb{R}^d\}_{k=1}^{m}$, with $\mathbf{P} = \bigcup_{i=1}^{L} \mathbf{P}_i$. The formulation of the deep-prompted model is as follows:

$$[\mathbf{x}_i, \_, \mathbf{E}_i] = B_i\big([\mathbf{x}_{i-1}, \mathbf{P}_{i-1}, \mathbf{E}_{i-1}]\big),\ i = 1, 2, \ldots, L. \tag{v}$$

where '_' indicates that VPT-Deep does not preserve the output corresponding to the prompt tokens $\mathbf{P}_{i-1}$.

Proximal Policy Optimization. PPO [48] is a widely used RL algorithm that can be applied to both discrete and continuous action spaces. Specifically, PPO-Clip updates policies by solving the following optimization problem:

$$\theta_{k+1} = \arg\min_{\theta}\ \mathbb{E}_{s,a\sim\pi_{\theta_k}}\big[L(s, a, \theta_k, \theta)\big], \tag{vi}$$

where π is the policy, θ is the policy parameters, and k is the k-th step. It typically takes multiple steps of SGD to optimize the clipped surrogate function $L(\cdot, \cdot, \cdot, \cdot)$:

$$L(s, a, \theta_k, \theta) = \min\Big(\frac{\pi_\theta(a\,|\,s)}{\pi_{\theta_k}(a\,|\,s)} A^{\pi_{\theta_k}}(s, a), \mathrm{clip}\Big(\frac{\pi_\theta(a\,|\,s)}{\pi_{\theta_k}(a\,|\,s)}, 1-\epsilon, 1+\epsilon\Big) A^{\pi_{\theta_k}}(s, a)\Big), \tag{vii}$$

where $A^{\pi_{\theta_k}}(s, a)$ is the advantage estimator, and ϵ is the clip hyper-parameter. The clipping mechanism constrains the new policy to stay close to the old one, preventing excessively large policy updates that could degrade performance [35, 48].

B. Pseudo-Code of PRO-VPT

We provide the detailed pseudo-code for PRO-VPT in Algo. i. Initially, it generates N prompts $\mathbf{P}$ and distributes them to the PVM according to a uniform distribution $\mathcal{D}$ (Line 1). In each epoch of PRO-VPT, it first calculates the estimated idleness scores $\{\widehat{\mathcal{I}}_k\}_{k=1}^{N}$ using Eq. (6) (Line 3). If $\max \widehat{\mathcal{I}}_k > 0$, the PR process is triggered. This process begins by pruning the prompt $\mathbf{p}_{k^*}$ with the highest score $\widehat{\mathcal{I}}_{k^*}$, thereby constructing an intermediate distribution $\mathcal{D}^-$ (Line 5). The current state of the distribution s is then computed, followed by determining the action a^* based on PPO (Lines 6 and 7). Subsequently, the pruned prompt $\mathbf{p}_{k^*}$ is allocated to the a^*-th block, resulting in a relocated distribution $\mathcal{D}^+$ (Line 8).

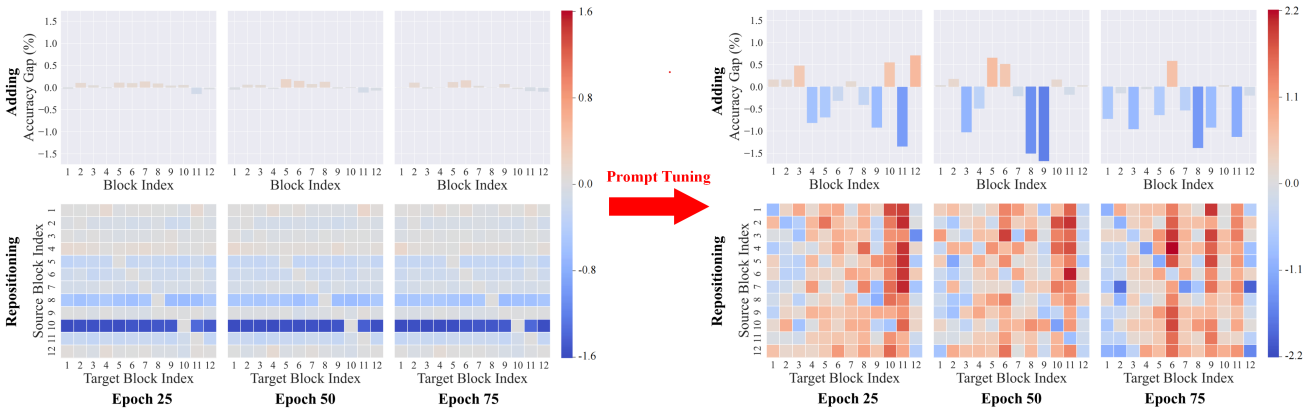

Figure i. **Detailed performance gaps from distribution adjustments** using prompts at epochs 25, 50, and 75, before and after prompt tuning. *Left:* Performance gaps from adjustments before prompt tuning. *Right:* Performance gaps from adjustments after prompt tuning. The effectiveness of distribution adjustments becomes apparent only after prompt tuning has been applied, and it also shifts with prompt updates.

Algorithm i: PRO-VPT

Input: pre-trained model f, number of epochs T, number of prompts N, learning rate η.

1 Initialize N prompts $\mathbf{P}$ and distribute them to the model according to a uniform distribution $\mathcal{D}$.
2 **for** $t = 0, \ldots, T-1$ **do**
3 Compute idleness scores $\{\widehat{\mathcal{I}}_k\}_{k=1}^{N}$ by Eq. (6).
4 **if** $\max \widehat{\mathcal{I}}_k > 0$ **then**
5 **Prune** the negative prompt $\mathbf{p}_{k^*}$ with the maximum idleness score $\widehat{\mathcal{I}}_{k^*}$ to form $\mathcal{D}^-$.
6 Compute the current state s.
7 Compute the action $a^* \leftarrow \text{PPO}(s)$.
8 **Allocate** the idle prompt $\mathbf{p}_{k^*}$ to the a^*-th block to form $\mathcal{D}^+$.
9 **end**
10 Update the prompts as $\mathbf{P}' \leftarrow \mathbf{P} - \eta \cdot \mathbf{g}$.
11 **if** $\max \widehat{\mathcal{I}}_k > 0$ **then**
12 Compute the reward $\hat{r}$ by Eq. (8).
13 Update the policy networks within PPO based on $\hat{r}$.
14 **end**
15 **end**

After completing the PR process, the prompts are optimized to $\mathbf{P}'$ (Line 10). Additionally, if the PR process is activated during the epoch, it is necessary to calculate the estimated reward $\hat{r}$ using Eq. (8) and update the policy networks in PPO accordingly (Lines 12 and 13).

C. Comprehensive Analysis of the Underlying Nature behind ADO

Fig. i presents the complete results of Fig. 2. Specifically, we investigated the performance gaps from distribution adjustments applied to prompts at different epochs, both before and after prompt tuning, on VTAB-1k *Natural* DTD using ViT-B/16. For the adding-based adjustment, we trained with a total of 59 prompts distributed uniformly across 12 Transformer blocks, resulting in one block containing 4 prompts while the others contained 5, then added a new prompt to the block with 4 prompts. For the repositioning-based adjustment, we trained with 60 prompts allocated uniformly, and then repositioned a single prompt. For fair comparisons, we averaged the results over five runs with different configurations, including variations in which block had the missing prompt as well as which specific prompt was repositioned.

The following provides a summary of the analysis and key findings presented in the main paper:

Finding 3. *The effectiveness of distribution adjustments becomes apparent only after prompt tuning, suggesting that these adjustments can only be properly evaluated after tuning, thereby forming a nested relationship between ADO and VPT.* Comparing the left and right parts of Fig. i, we find that adjusting the distribution without prompt tuning leads to negligible changes or even degrades performance. In contrast, implementing prompt tuning after adjusting the distribution leads to significant performance changes, with well-chosen adjustments leading to notable improvements (*e.g.*, an increase of up to 2.2 pp achieved by repositioning just a single prompt). To this end, the proper workflow for ADO and VPT should be established as 'distribution adjustment $\rightarrow$ prompt tuning $\rightarrow$ adjustment evaluation', with prompt tuning nested within the distribution optimization process.

Finding 2. *Adjustments in prompt distribution are influenced by the updated prompts themselves, underscoring the necessity of an iterative process that continuously refines the distribution over prompt updates.* As demonstrated in the right portion of Fig. i, for prompts from epoch 25, we observe significant performance gains by adding a new prompt to the 10-th and 12-th blocks, as well as by repositioning one

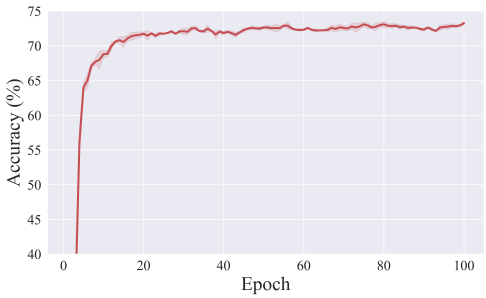

Figure ii. **Convergence curve across different prompt-tuning epochs**, which corresponds to Fig. i.

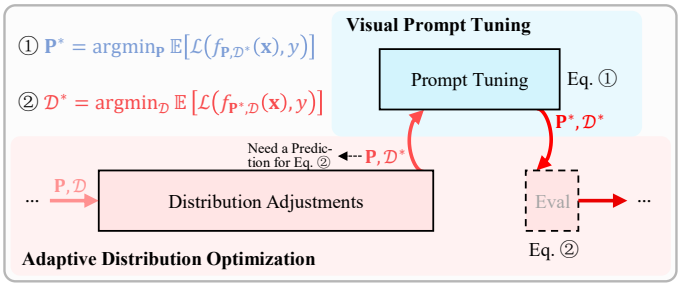

Figure iii. **Proper workflow of ADO-VPT co-design framework.** It is framed as an iterative process, with the VPT process nested within the ADO process.

prompt to the 10-th and 11-th blocks. However, for prompts at epoch 75, the key improvements shifted to adding to the 6-th block and repositioning to the 6-th and 9-th blocks. To this end, the ADO-VPT co-design workflow should be established as an iterative process, which would be 'distribution adjustment → prompt tuning → adjustment evaluation → new adjustment'.

Additionally, to validate that the performance improvements from applying both distribution adjustments and prompt tuning mainly result from the distribution adjustments rather than the tuning itself, we present the respective prompt tuning accuracy curve in Fig. ii. Clearly, the prompted model has already converged around epoch 20, with only slight performance fluctuations thereafter (less than 0.3 pp). This indicates that the performance differences observed in Fig. i (typically exceeding 0.3 pp) cannot be attributed solely to prompt tuning alone, but rather to the adjustments for the prompt distribution.

Building upon the underlying nature described in **Findings 2** and **3**, a proper workflow for ADO and VPT should be structured as an iterative nested process. Specifically, in each iteration, the process first adjusts the distribution to construct $\mathcal{D}^*$, then tunes the visual prompts to obtain $\mathbf{P}^*$. Based on $\mathcal{D}^*$ and $\mathbf{P}^*$, the effectiveness of the distribution adjustment is evaluated at the end of each iteration, marking the completion of the current cycle and the beginning of the next. Fig. iii illustrates this co-design workflow for ADO and VPT. Formally, it can be expressed as a nested optimization problem as follows:

$$\mathcal{D}^* = \arg\min_{\mathcal{D}} \mathbb{E}_{(\mathbf{x},y)\in\mathcal{T}_{tr}}\left[\mathcal{L}(f_{\mathbf{P}^*,\mathcal{D}}(\mathbf{x}),y)\right], \quad \text{(viii)}$$

$$\mathbf{P}^* = \arg\min_{\mathbf{P}} \mathbb{E}_{(\mathbf{x},y)\in\mathcal{T}_{tr}}\left[\mathcal{L}(f_{\mathbf{P},\mathcal{D}^*}(\mathbf{x}),y)\right], \quad \text{(ix)}$$

where the notations $\mathbf{P}^*$ and $\mathcal{D}^*$ correspond to those depicted in Fig. iii.

Overall, the ADO problem is naturally formulated as a nested problem, grounded in its underlying nature. This formulation is applicable to various distribution adjustment strategies (*e.g.*, adding, repositioning, or even pruning) as well as different discrete optimization methods (*e.g.*, evolutionary algorithms and reinforcement learning). Notably, since distribution adjustments can only be evaluated at the end, a prediction for Eq. viii is necessary for selecting an appropriate adjustment. To this end, it is natural to frame ADO as a RL problem, where the reward is predicted to determine the next action, and the effectiveness of that action is evaluated afterward.

D. Inferior Performance of Adding-based Adjustments

Following the nested optimization framework described in Eqs. (viii) and (ix), we have also attempted to develop an adding-based strategy for ADO. Given that framing the ADO objective as a RL problem is a suitable choice (refer to § 3.3 and § C), we also address the adding-based ADO by leveraging RL, with the components of the Markov decision process specified as follows:

1) State. We utilize the current prompt distribution as the state, denoted as $s = \mathcal{D}$. After adding a new prompt and performing prompt tuning, the state transitions to s'.

2) Action. Unlike prompt repositioning, which considers L^2 possible block arrangements, the adding-based ADO involves only L potential adding operations. As a result, the adding-based ADO does not require decomposition for RL,

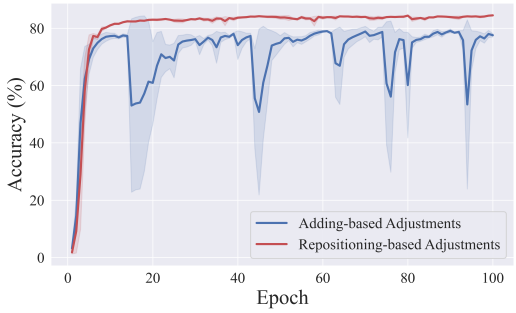

Figure iv. **Convergence curves comparing adding-based and repositioning-based strategies.** The adding-based strategy exhibits significantly less stability.

and the action is straightforwardly represented as $a \in [L]$.

3) Reward. Similar to the PR strategy, the reward is formulated based on Eq. (viii) as $r = \Delta\mathcal{L}(f_{\mathbf{P},\mathcal{D}}, f_{\mathbf{P}',\mathcal{D}'})$, where $\mathcal{D}'$ denotes the updated prompt distribution after adding and $\mathbf{P}'$ represents the tuned prompts.

Similarly, we employ PPO for this RL problem to tackle the adding-based ADO objective, while adopting the overall framework illustrated in Fig. iii.

Fig. iv illustrates the performance comparison between the adding-based strategy and repositioning-based strategy (PRO-VPT) on the VTAB-1k *Natural* Cifar100 dataset. It can be clearly observed that the adding-based strategy is significantly less stable and underperforms compared to the repositioning-based approach. We attribute this discrepancy to the undertraining of newly added prompts and potential conflicts with existing ones. Consequently, our work focuses on developing the repositioning-based strategy for ADO.

E. Relocating One Prompt for Each Iteration

Here, we explain why we restrict the relocation process to operate on only a single prompt for each iteration. Since we frame the allocation step as a RL problem, we need to evaluate the effectiveness of each allocation decision for relocated prompts. Relocating more than one prompt simultaneously would necessitate multiple reward evaluations, substantially increasing task complexity and computational overhead. Therefore, we designed to relocate only one prompt per iteration to maintain simplicity and efficiency in the allocation process.

F. Derivation of the Taylor Expansion for the Idleness Score

Based on the pruning objective in Eq. (3), the original idleness score is defined as:

$$\mathcal{I}_k = \Delta\mathcal{L}(f_{\mathbf{P},\mathcal{D}}, f_{\mathbf{P},\mathcal{D}|d_k=0}), \tag{x}$$

where $d_k = 0$ indicates that the k-th prompt is pruned. Equivalently, this equation can be expressed as:

$$\mathcal{I}_k = \Delta\mathcal{L}(f_{\mathbf{P},\mathcal{D}}, f_{\mathbf{P}|\mathbf{p}_k=\mathbf{0},\mathcal{D}}), \tag{xi}$$

where $\mathbf{p}_k = \mathbf{0}$ represents that the k-th prompt is a zero vector.

Let $p_{kj} \in \mathbf{p}_k$ denote as the prompt parameter. The difference in losses with and without the prompt parameter, *i.e.*, the idleness score of p_{kj}, is given by:

$$\mathcal{I}_{kj} = \Delta\mathcal{L}(f_{\mathbf{P},\mathcal{D}}, f_{\mathbf{P}|p_{kj}=0,\mathcal{D}}). \tag{xii}$$

Considering the entire prompt set as a concatenated vector $\mathbf{P} = \{p_{00}, p_{01}, \ldots, p_{Nd}\}$, we are able to approximate $\mathcal{I}_{kj}$ in the vicinity of $\mathbf{P}$ by its first-order Taylor expansion:

$$\begin{aligned}\widehat{\mathcal{I}}_{kj} &\approx \mathbf{g}^T\Big(\mathbf{P} - \mathbf{P}\,|\,p_{kj}=0\Big) \\ &\approx g_{kj}p_{kj},\end{aligned} \tag{xiii}$$

where $g_{kj} = \frac{\nabla\mathcal{L}}{\nabla p_{kj}}$ represents the element of the gradient $\mathbf{g}$.

For the idleness score of a prompt $\mathbf{p}_k = \{p_{kj}\}_{j=1}^d$, we can approximate it by summing the score of its individual parameters, as follows:

$$\widehat{\mathcal{I}}_k \approx \sum_{j=1}^{d} g_{kj}p_{kj} \approx \mathbf{g}_k^T\mathbf{p}_k. \tag{xiv}$$

To this end, we are able to efficiently approximate the idleness scores $\{\mathcal{I}_k\}_{k=1}^N$ by a single backpropagation pass, thereby avoiding the need to evaluate the idleness scores of all N prompts individually.

Table i. **Hyper-parameters for VPT and PRO-VPT.**

	VPT	PRO-VPT
Batch size	64 ($p \geq 100$), 128 ($p < 100$)	64
Learning rate schedule	cosine decay	-
Optimizer	SGD	
Optimizer momentum	0.9	
$base\_lr$ range	{50., 25., 10., 5., 2.5, 1., 0.5, 0.25, 0.1, 0.05}	
Weight decay range	{0.01, 0.001, 0.0001, 0.0}	
Drop rate	0.1	
Total epochs	100	

G. More Implementation Details

Data Augmentation. Apart from data normalization, we resize the input images to 224×224 pixels for VTAB-1k and apply a randomly resize crop to 224×224 pixels and horizontal flipping for FGVC, as outlined in [13, 22, 51, 59].

Training Hyper-Parameters. Specific to training hyper-parameters, we largely adopt the same settings as depicted in VPT [22]. Tab. i summarizes the hyper-parameter configurations comparing the experiments of VPT and our approach. Following [13, 22], we conduct a grid search on the validation set of each task to determine the optimal learning rate and weight decay; the learning rate is set as $base\_lr \times b/256$, where b denotes the batch size and $base\_lr$ is selected from the range specified in Tab. i. Notably, PRO-VPT does not require specific-designed large learning rates as in [22]. For all experiments conducted with our implementation, the results are averaged over three random seeds.

PPO Hyper-Parameters. We also detail the hyper-parameters of our PPO implementation for reproducibility. Both the actor and critic networks are two-layer MLPs with 64 hidden units per layer. The total number of parameters of the two policy networks is precisely 0.0136M. The learning rates are 0.0003 for the actor and 0.001 for the critic. Additionally, we set the discount factor to 1 and the clipping factor to 0.2.

Table ii. **Specifications of the VTAB-1k datasets.**

Group	Task	# Classes	Splits		
			Train	Val	Test
Natural	CIFAR-100	100	800	200	10 000
	Caltech-101	102			6 084
	DTD	47			1 880
	Oxford Flowers	102			6 149
	Pets	37			3 669
	SVHN	10			26 032
	Sun397	397			21 750
Specialized	Patch Camelyon	2	800	200	32 768
	EuroSAT	10			5 400
	RESISC45	45			6 300
	Diabetic Retinopathy	5			42 670
Structured	CLEVR-Count	8	800	200	15 000
	CLEVR-Distance	6			15 000
	DMLab	6			22 735
	KITTI-Distance	4			711
	dSprites-Location	16			73 728
	dSprites-Orientation	16			73 728
	smallNORB-Azimuth	18			12 150
	smallNORB-Elevation	9			12 150

Datasets and Pre-Trained Backbones Specifications. Tabs. ii and iii present the statistics of each task in VTAB-1k and FGVC *w.r.t.* the number of classes and the number of images in the train, validation, and test splits. The tables are largely "borrowed" from [51]. Moreover, Tab. iv provides the details of the pre-trained backbones used in this paper, which is largely "borrowed" from [22].

Reproducibility. PRO-VPT is implemented in Pytorch and timm. Experiments are conducted on NVIDIA A30-24GB GPUs. To guarantee reproducibility, our full implementation will be publicly released.

H. More Experimental Results

Complete Results for VTAB-1k. Tab. v presents comprehensive results for VTAB-1k, using both ImageNet and Inception normalizations. Although several of the best pre-task results from other PEFT methods differ and exceed those listed in Tab. 1, PRO-VPT remains highly competitive, achieving a state-of-the-art average accuracy of 78.0% among all evaluated methods.

Fig. v illustrates a comparison of prompt-based methods, including the previous state-of-the-art (iVPT), the baseline (VPT), and our proposal (PRO-VPT). All scores were normalized by $x_{\text{norm}} = x - x_{\text{mean}}$. It is evident that PRO-VPT outperforms both current leading methods, establishing a new state-of-the-art performance for prompt-based techniques.

Complete Results for FGVC. Tab. vi presents comprehensive results for FGVC, utilizing both ImageNet and Inception normalizations. The best pre-task results remain consistent with those in Tab. 2, and PRO-VPT demonstrates superior performance on large-scale fine-grained datasets.

Table iii. **Specifications of the FGVC datasets.** For datasets marked with *, we follow [51] to randomly sample train and validation splits since validation sets are not available from the original datasets.

Dataset	# Classes	Splits		
		Train	Val	Test
CUB-200-2011* [56]	200	5 394	600	5 794
NABirds* [55]	555	21 536	2 393	6 084
Oxford Flowers [42]	102	1 020	1 020	6 149
Stanford Dogs* [6]	120	10 800	1 200	8 580
Stanford Cars* [10]	196	7 329	815	8 041

Table iv. **Specifications of the pre-trained backbones.**

Backbone	Pre-trained Strategy	Pre-trained Dataset	Param (M)	Feature dim d	Pre-trained Model
ViT-B/16	Supervised	ImageNet-21k	85	768	checkpoint
ViT-L/16			307	1024	checkpoint
ViT-H/14			630	1280	checkpoint
Swin-B	Supervised	ImageNet-21k	88	1024	checkpoint
ViT-B/16	MAE	ImageNet-1k	85	768	checkpoint
ViT-B/16	MoCo-v3				checkpoint

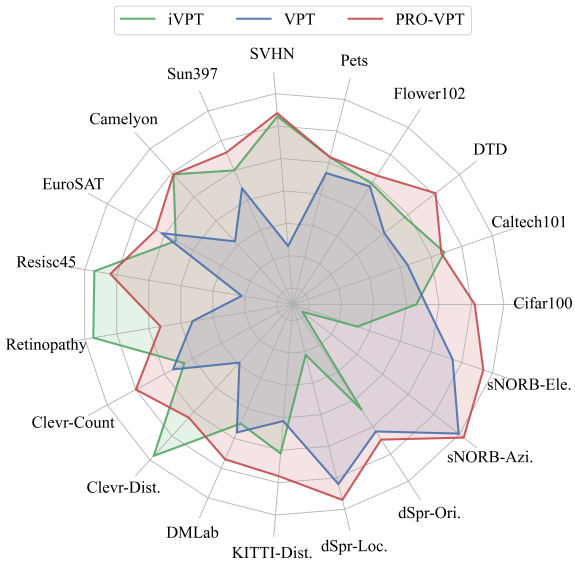

Figure v. **Comparison of prior state-of-the-art (iVPT), baseline (VPT), and our method (PRO-VPT).** Our approach subsumes two representative methods.

Generalizability Study on Detection and Segmentation Tasks. We also conduct experiments on a broader range of downstream tasks, including object detection, instance segmentation, and semantic segmentation. Specifically, we evaluate object detection and instance segmentation performance on the COCO dataset [36], using Mask R-CNN [17] with a Swin-T backbone pre-trained on ImageNet-1k. For the

Table v. **Comprehensive results on the VTAB-1k datasets.** Performance results are reported using both ImageNet normalization (○) or Inception normalization (●), presented in % after a complete training schedule with ViT-B/16 supervised pre-trained on ImageNet-21k. The best results of prompt-based methods and other PEFT approaches are highlighted in **bold**. ↯: Early-stopping based on the test set. †: Lack of complete code or hyperparameter configurations for the method, hence results are reported as presented in the original paper. [1]Average across the average accuracies of the VTAB-1k groups, following previous work.

		Natural								Specialized					Structured										
	Param (M)	Cifar100	Caltech101	DTD	Flower102	Pets	SVHN	Sun397	Group Avg.	Camelyon	EuroSAT	Resisc45	Retinopathy	Group Avg.	Clevr-Count	Clevr-Dist.	DMLab	KITTI-Dist.	dSpr-Loc.	dSpr-Ori.	sNORB-Azi.	sNORB-Ele.	Group Avg.	Global Avg.[1]
Full ○	85.8	68.9	87.7	64.3	97.2	86.9	87.4	38.8	75.9	79.7	95.7	84.2	73.9	83.4	56.3	58.6	41.7	65.5	57.5	46.7	25.7	29.1	47.6	69.0
Full ●	85.8	73.2	92.6	70.4	97.9	86.2	90.6	39.6	78.6	87.1	96.6	87.5	74.0	86.3	66.6	61.0	49.8	79.7	82.6	51.9	33.5	37.0	57.8	74.2
Linear ○	0.04	63.4	85.0	63.2	97.0	86.3	36.6	51.0	68.9	78.5	87.5	68.6	74.0	77.2	34.3	30.6	33.2	55.4	12.5	20.0	9.6	19.2	26.9	57.7
Linear ●	0.04	78.1	88.1	69.0	99.1	90.0	36.0	56.9	73.9	79.8	90.7	73.7	73.7	79.5	32.4	30.5	35.9	61.9	11.2	26.2	14.3	24.5	29.6	61.0
LoRA ● [20]	0.29	83.0	91.7	71.6	99.2	90.9	83.8	56.7	82.4	86.2	95.7	83.5	71.9	84.3	77.7	62.3	49.0	80.2	82.2	51.7	31.0	47.0	60.1	75.6
FacT-TK$_8$ ○ [23]	**0.05**	70.3	88.7	69.8	99.0	90.4	84.2	53.5	79.4	82.8	95.6	82.8	75.7	84.2	81.1	68.0	48.0	80.5	74.6	44.0	29.2	41.1	58.3	74.0
FacT-TK$_8$ ● [23]	**0.05**	74.9	92.7	73.7	99.1	91.3	85.5	**57.7**	82.1	86.8	94.9	84.1	70.9	84.2	81.9	64.1	49.2	77.2	83.8	53.1	28.2	44.7	60.3	75.5
FacT-TK$_{\leq 32}$ ○ [23]	0.10	70.6	90.6	70.8	99.1	90.7	88.6	54.1	80.6	84.8	96.2	84.5	75.7	85.3	82.6	**68.2**	49.8	80.7	80.8	47.4	33.2	43.0	60.7	75.6
FacT-TK$_{\leq 32}$ ● [23]	0.10	74.6	93.7	73.6	99.3	90.6	88.7	57.5	82.6	**87.6**	95.4	85.5	70.4	84.7	**84.3**	62.6	51.9	79.2	85.5	52.0	36.4	46.6	62.3	76.5
Consolidator † [15]	0.30	74.2	90.9	73.9	**99.4**	91.6	**91.5**	55.5	82.4	86.9	95.7	86.6	**75.9**	86.3	81.2	**68.2**	51.6	**83.5**	79.8	52.3	31.9	38.5	60.9	76.5
SSF ↯○ [34]	0.24	69.0	92.6	**75.1**	**99.4**	91.8	90.2	52.9	81.6	87.4	95.9	87.4	75.5	**86.6**	75.9	62.3	53.3	80.6	77.3	54.9	29.5	37.9	59.0	75.7
SSF ↯● [34]	0.24	61.9	92.3	73.4	**99.4**	**92.0**	90.8	52.0	80.3	86.5	95.8	**87.5**	72.8	85.7	77.4	57.6	53.4	77.0	78.2	54.3	30.3	36.1	58.0	74.6
SPT-Adapter ↯○ [16]	0.23	72.9	93.2	72.5	99.3	91.4	84.6	55.2	81.3	85.3	96.0	84.3	75.5	85.3	82.2	68.0	49.3	80.0	82.4	51.9	31.7	41.2	60.8	75.8
SPT-Adapter ↯● [16]	0.22	74.7	94.1	73.0	99.1	91.2	84.5	57.5	82.0	85.7	94.9	85.7	70.2	84.1	81.3	63.2	49.1	80.7	83.5	52.0	26.4	41.5	59.7	75.3
SPT-Adapter ↯○ [16]	0.43	72.9	93.2	72.5	99.3	91.4	88.8	55.8	82.0	86.2	96.1	85.5	75.5	85.8	83.0	68.0	51.9	81.2	82.4	51.9	31.7	41.2	61.4	76.4
SPT-Adapter ↯● [16]	0.43	74.9	93.2	71.6	99.2	91.1	87.9	57.2	82.2	87.0	95.4	86.5	72.4	85.3	81.1	63.2	50.3	80.2	84.4	51.4	31.5	42.2	60.5	76.0
Adapter+$_{r=16}$ ● [51]	0.35	**83.7**	**94.2**	71.5	99.3	90.6	88.2	55.8	**83.3**	87.5	**97.0**	87.4	72.9	86.2	82.9	60.9	**53.7**	80.8	**88.4**	**55.2**	**37.3**	**46.9**	**63.3**	**77.6**
Prompt-based Methods:																								
VPT-Deep ○ [22]	0.60	78.8	90.8	65.8	98.0	88.3	78.1	49.6	78.5	81.8	96.1	83.4	68.4	82.4	68.5	60.0	46.5	72.8	73.6	47.9	32.9	37.8	55.0	72.0
VPT-Deep ● [22]	0.60	83.0	93.0	71.2	99.0	91.3	84.1	56.0	82.5	84.9	96.6	82.5	74.5	84.6	77.5	58.7	49.7	79.6	86.2	56.1	37.9	50.7	62.1	76.4
NOAH ↯● [67]	0.43	69.6	92.7	70.2	99.1	90.4	86.1	53.7	80.2	84.4	95.4	83.9	75.8	84.9	**82.8**	**68.9**	49.9	**81.7**	81.8	48.3	32.8	44.2	61.3	75.5
SPT-Deep † [58]	**0.22**	79.3	92.6	**73.2**	**99.5**	91.0	89.1	51.2	82.3	85.4	**96.8**	84.9	74.8	85.5	70.3	64.8	**54.2**	75.2	79.3	49.5	36.5	41.5	58.9	75.6
iVPT † [71]	0.60	82.7	**94.2**	72.0	99.1	**91.8**	88.1	56.6	83.5	**87.7**	96.1	**87.1**	**77.6**	**87.1**	77.1	62.6	49.4	80.6	82.1	55.3	31.8	47.6	60.8	77.1
PRO-VPT (**ours**)	0.61	**84.5**	94.1	**73.2**	99.4	**91.8**	**88.2**	**57.2**	**84.1**	**87.7**	**96.8**	86.6	75.5	86.7	78.8	61.0	50.6	81.3	**86.7**	**56.4**	**38.1**	**51.7**	**63.1**	**78.0**

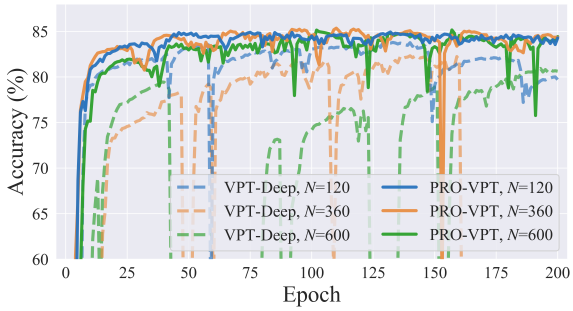

Figure vi. **Convergence curves comparing VPT and PRO-VPT** with varying numbers of prompts. PRO-VPT exhibits superior robustness.

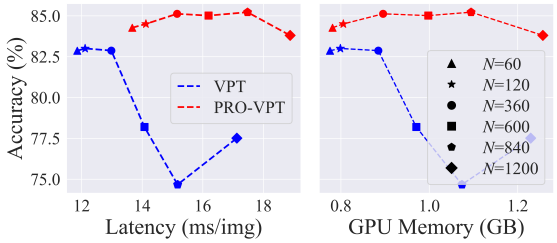

Figure vii. **Cost comparisons** with different prompt numbers. We report the latency (ms/img) and the GPU memory usage (GB). The increased training time associated with PRO-VPT is justified by the considerable enhancements in both performance and robustness.

semantic segmentation task, we use the ADE20k dataset [70] and adopt SETR-PUP [69] with a ViT-B/16 backbone pre-trained on ImageNet-21k. As shown in Tab. vii, the results demonstrate that PRO-VPT consistently outperforms VPT across both detection and segmentation tasks, highlighting its superior generalizability.

Convergence Curves with Different Prompt Numbers. Fig. vi illustrates the convergence curves under varying total numbers of prompts on VTAB-1k *Natural* Cifar100, with training extended to 200 epochs. Notably, VPT exhibits high sensitivity to the number of prompts, as discussed in [21, 58]. An improper prompt number can lead to significant instability during convergence and result in inferior performance. In contrast, PRO-VPT demonstrates remarkable robustness to variations in prompt quantity. Although it is slightly affected during the initial convergence phase, our method ultimately converges to consistent performance. This suggests that learning the optimal distribution enables a more nuanced calibration of tuning intensity at each block, thereby enhancing robustness.

Cost Analysis. As detailed in § 3, estimating the expected objectives effectively avoids additional computation, thereby

Table vi. **Comprehensive results on the FGVC datasets.** Performance results are reported as the highest of ImageNet normalization (○) or Inception normalization (●), presented in % after a complete training schedule with ViT-B/16 supervised pre-trained on ImageNet-21k. The best results of prompt-based methods and other PEFT approaches are highlighted in **bold**. †: Lack of complete code or hyperparameter configurations for the method, hence results are reported as presented in the original paper.

	Param (M)	CUB200	NABirds	Oxford Flowers	Stanford Dogs	Stanford Cars	Global Avg.
Full ○	86.0	87.3	82.7	98.8	89.4	84.5	88.5
Full ●	86.0	88.0	81.5	99.2	85.6	90.6	89.0
Linear ○	0.18	85.3	75.9	97.9	86.2	51.3	79.3
Linear ●	0.18	88.9	81.8	99.5	92.6	52.8	83.1
SSF ○ [34]	0.39	89.5	**85.7**	99.6	89.6	**89.2**	90.7
SSF ● [34]	0.39	88.9	85.0	99.6	88.9	88.9	90.3
SPT-Adapter † [16]	0.40	89.1	83.3	99.2	91.1	86.2	89.8
SPT-LoRA † [16]	0.52	88.6	83.4	99.5	91.4	87.3	90.1
Adapter+ ● [51]	**0.34**	**90.4**	85.0	**99.7**	**92.6**	89.1	**91.4**
Prompt-based Methods:							
VPT-Deep ○ [22]	0.85	88.5	84.2	99.0	90.2	83.6	89.1
VPT-Deep ● [22]	0.85	90.1	83.3	99.6	90.3	85.0	89.7
SPT-Deep † [58]	**0.36**	**90.6**	**87.6**	**99.8**	89.8	89.2	91.4
iVPT † [71]	0.41	89.1	84.5	99.5	90.8	85.6	89.9
PRO-VPT (**ours**)	0.86	**90.6**	86.7	99.7	**91.8**	**89.6**	**91.7**

Table vii. **Generalizability study on detection and segmentation tasks.** Results are presented on two instances: COCO val2017 and ADE20k.

	COCO with Mask R-CNN						ADE20k with SETR	
	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}	mIoU-SS	mIoU-MS
VPT-Deep	33.8	57.6	35.3	32.5	54.5	33.9	39.1	40.1
PRO-VPT	**34.6**	**58.6**	**36.1**	**33.4**	**55.5**	**34.7**	**40.0**	**41.0**

[1] AP^b and AP^m are the average precision for objective detection and instance segmentation.
[2] mIoU-SS and mIoU-MS are single- and multi-scale inference of semantic segmentation.

significantly enhancing the efficiency of our method. In particular, the extra training cost of our method primarily stems from the computation of policy networks in PPO. However, since the policy networks are merely lightweight MLPs, this additional cost is relatively low. Specifically on VTAB-1k, the average latency for VPT and PRO-VPT is 13.37 ms/img and 15.55 ms/img (**1.16**×). Furthermore, we evaluate latency and GPU usage with varying prompt numbers on VTAB-1k *Natural* Cifar100 in Fig. vii. Although PRO-VPT does introduce some extra training time, the performance and robustness improvements justify this cost, and the increase in GPU memory usage is marginal.

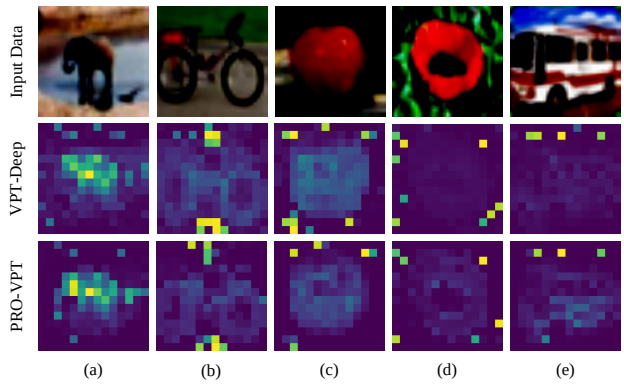

Figure viii. **Visualization of attention maps**. PRO-VPT exhibits more focused and precise attention with fewer artifacts.

I. Visualization and Analysis

Attention Maps. We visualize the attention maps between [CLS] and image patches on VTAB-1k *Natural* Cifar100. As shown in Figs. viii(a)-(c), while VPT successfully focuses on the object, its attention exhibits significant artifacts [4] and, more critically, remains scattered. For example, in Fig. viii(a), VPT shows certain attention on the lake, which is actually part of the background. In contrast, PRO-VPT demonstrates more focused and accurate attention with fewer artifacts. Furthermore, as illustrated in Figs. viii(d) and (e), VPT appears to struggle with effectively concentrating on the object, whereas PRO-VPT maintains its ability to focus on the object.

Learned Distributions and Accuracy Curves. Fig. ix illustrates the learned distributions in PRO-VPT as well as the accuracy curves in comparison to VPT, based on 100 training epochs. Empirical results from more datasets further reinforce that the prompting importance for each block is inherently task-dependent. Moreover, the comparison of accuracy curves between PRO-VPT and VPT, particularly on the SVHN, Camelyon, and Resisc45 datasets, reveals that PRO-VPT still exhibits an upward trend in accuracy during the late training stages. This highlights the effectiveness of learning the optimal distribution for visual prompts, which unlocks their full potential and maximizes downstream performance.

J. Limitation and Future Work

Despite demonstrating promising performance and enhanced robustness, our approach still has certain limitations. First, as noted in [21, 58], prompt-based methods are significantly sensitive to the total number of prompts. Although our method improves robustness and mitigates this issue to some extent (refer to § 4.5), there is still room for further refinement. In particular, while PRO-VPT maintains robust performance within a certain range (*e.g.*, 120-600 initialized

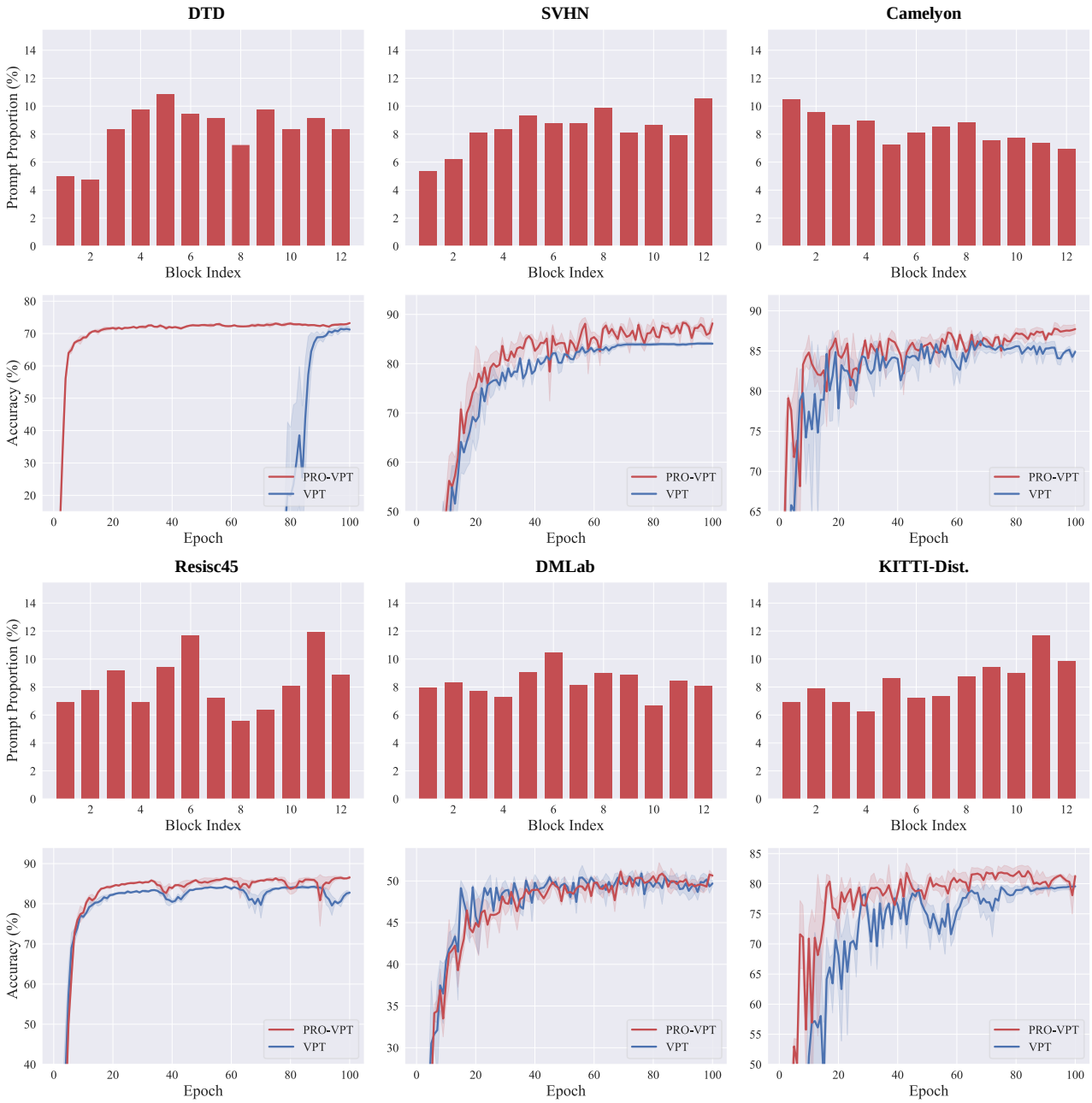

Figure ix. **Visualization of the prompt distributions learned by PRO-VPT and the accuracy curves compared to VPT** on the VTAB-1k datasets: *Natural* DTD, SVHN; *Specialized* Camelyon, Resisc45; and *Structured* DMLab, KITTI-Dist.

prompts as in Fig. vi), its performance still deteriorates when the number of prompts deviates significantly from the optimal value. Consequently, our method still employs the optimal total number of prompts per task from [22], resulting in relatively high parameter counts as in VPT. Second, the constraint of relocating one prompt per iteration (see § E) restricts our method to relocating only limited prompts within certain epochs. This particularly affects tasks requiring high prompt numbers, hindering PRO-VPT's ability to learn optimal distributions. For instance, the changes from initial uniform to learned prompt distributions are relatively less pronounced for Camelyon and DMLab as shown in Fig. ix, which require 100 initialized prompts per block. Nevertheless, it is worth noting that PRO-VPT still delivers considerable performance enhancements on these datasets. Addressing these limitations would further enhance the efficiency and applicability of our approach in diverse downstream tasks.

Moreover, while this paper focuses on calibrating the distribution of prompts, the PRO-VPT framework is applicable to most block-wise PEFT approaches, *e.g.*, adjusting the pre-block dimension of subspaces (*i.e.*, the rank distribution) in adapter-based methods. An essential future direction deserving of further investigation is integrating our approach with other PEFT methods, developing a universal framework for fine-grained robust PEFT.