Appendix

We provide details omitted in the main paper.

- Appendix A: a conceptual comparison of VQT with other transfer learning methods.
- Appendix B: additional experiment details (cf. section 4 of the main paper).
- Appendix C: additional experiment results and analyses (cf. section 4 of the main paper).
- Appendix D: additional discussions.

A. Conceptual Comparison between VQT and Transfer Learning Methods

We demonstrate the conceptual difference between various transfer learning methods in Figure 7. Figure 7a shows the pre-trained ViT backbone. Linear-probing (Figure 7b) only updates the prediction head and keeps the rest of the backbone unchanged, while fine-tuning (Figure 7c) updates the whole backbone. HEAD2TOE (Figure 7d) takes intermediate outputs of blocks within each Transformer layer for predictions. In contrast, our VQT (Figure 7e) leverages the summarized intermediate features of each Transformer layer for final predictions.

B. Additional Experiment Details

B.1. Pre-training Setups

In subsection 4.2 and subsection 4.3 of the main paper, we conduct our experiments on three types of pre-trained backbones, Supervised [13], MAE [20], and CLIP [39]. We briefly review these pre-training methods in the following.

Supervised. Given a pre-training data set $\mathcal{D}_{\text{pre-train}} = \{(\boldsymbol{I}_i, y_i)\}_{i=0}^{N}$, where $\boldsymbol{I}_i$ is an image and $y_i \in [C]$ is the annotated class label, we aim to train a network that classifies images into C classes. The network consists of a backbone network f to extract features and a linear classifier h to predict class labels. Specifically, let $\boldsymbol{p}_i = h(f(\boldsymbol{I}_i)) \in \mathbb{R}^C$ be the output of the whole network. Each element of $\boldsymbol{p}_i$ represents the score of $\boldsymbol{I}_i$ being classified to each of the C classes. We apply the standard cross-entropy loss to maximize the score of classifying $\boldsymbol{I}_i$ to be the class y_i. After pre-training, we discard the classifier h and only keep the pre-trained backbone f for learning downstream tasks.

In our experiments, we use ViT-B/16 as the backbone architecture. In subsection 4.2, we use the ImageNet-1K pre-trained backbone following HEAD2TOE [15]. In subsection 4.3, we use the ImageNet-21K pre-trained backbone following VPT [23]. To save the pre-training time, we use the checkpoints of these backbones released on the official GitHub page of Vision Transformer [13][4].

MAE. The learning objective of MAE is to reconstruct an image from its partial observation. Specifically, we divide an input image $\boldsymbol{I}$ into N fixed-sized non-overlapping patches $\{\boldsymbol{I}^{(n)}\}_{n=1}^{N}$ following ViT [13]. Then, we randomly mask $K\%$ of the patches. Let $\mathcal{U}$ be the set of indices of the unmasked patches and $|\mathcal{U}| = (1 - K\%) \times N$. The goal is to reconstruct the masked patches using the unmasked ones $\{\boldsymbol{I}^{(i)} | i \in \mathcal{U}\}$. To achieve this, we first process the unmasked patches by a ViT encoder f to generate the output $\boldsymbol{Z}_M = [\boldsymbol{x}_M^{(i_1)}, \cdots, \boldsymbol{x}_M^{(i_{|\mathcal{U}|})}]$, where $i_1, \cdots, i_{|\mathcal{U}|} \in \mathcal{U}$. Then, we expand $\boldsymbol{Z}_M$ to have N tokens by filling $K\% \times N$ mask tokens $\boldsymbol{x}^{(\text{Mask})}$ into the positions of the masked patches to generate $\tilde{\boldsymbol{Z}}_M$. The mask token $\boldsymbol{x}^{(\text{Mask})}$ is a learnable parameter and indicates the missing patches to be predicted. Finally, we use a decoder h to generate the reconstructed image $\tilde{\boldsymbol{I}} = h(\tilde{\boldsymbol{Z}}_M)$. The whole encoder-decoder network is trained by comparing $\tilde{\boldsymbol{I}}$ with $\boldsymbol{I}$ by using the mean squared error (MSE). Similar to BERT [25], the loss is computed only on the masked patches. After pre-training, we discard the decoder h and only keep the encoder f as the backbone. In subsection 4.3, we use the ViT-B/16 backbone released in the official MAE [20] GitHub page[5].

CLIP. CLIP leverages text captions of images as supervisions for pre-training a visual model. The learning objective is to predict which text caption is paired with which image within a batch. Specifically, given a batch of image-caption pairs $\{(\boldsymbol{I}_i, \boldsymbol{C}_i)\}_{i=1}^{B}$, where $\boldsymbol{I}_i$ is an image and $\boldsymbol{C}_i$ is the caption, CLIP uses a image encoder f and a text encoder h to map $\boldsymbol{I}_i$ and $\boldsymbol{C}_i$ into a multi-modal embedding space, respectively. Let $\boldsymbol{Z}_I = [f(\boldsymbol{I}_1), \cdots, f(\boldsymbol{I}_B)] \in \mathbb{R}^{D\times B}$ and $\boldsymbol{Z}_C = [h(\boldsymbol{C}_1), \cdots, h(\boldsymbol{C}_B)] \in \mathbb{R}^{D\times B}$ be the output image and text features. We then compute pair-wise similarity between the columns of $\boldsymbol{Z}_I$ and $\boldsymbol{Z}_C$, resulting in $\boldsymbol{S} = \boldsymbol{Z}_I^T \boldsymbol{Z}_C \in \mathbb{R}^{B\times B}$. The diagonal elements in $\boldsymbol{S}$ are the scores for the correct image-caption pairings while the rest elements are incorrect pairings. CLIP minimizes the cross-entropy losses computed on the rows and the columns of $\boldsymbol{S}$ to learn f and h. After pre-training, we discard h and keep the vision encoder f as the pre-trained backbone. In subsection 4.3, we use the ViT-B/16 backbone released in the official CLIP [39] GitHub page[6].

B.2. Feature Selection via Group Lasso

We provide more details for feature selection based on group lasso as mentioned in subsection 3.2 of the main pa-

[4]https://github.com/google-research/vision_transformer

[5]https://github.com/facebookresearch/mae

[6]https://github.com/openai/CLIP

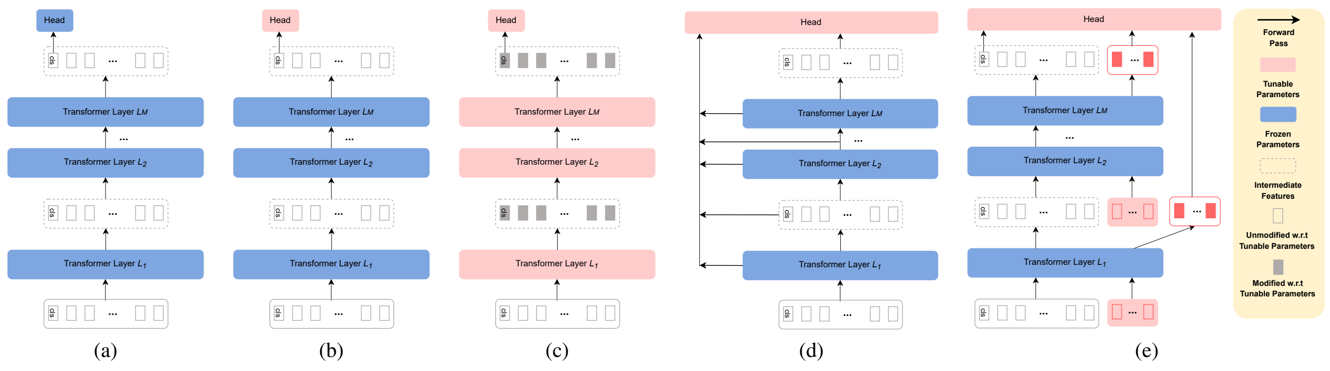

Figure 7. **Conceptual comparison between different transfer learning methods** (a) ViT Backbone. (b) Linear-Probing. (c) Fine-Tuning. (d) Head2Toe [15]. (d) VQT (Ours).

per. In VQT, we concatenate the newly summarized features $\{Z'_{m+1}\}_{m=0}^{M-1}$ with the final "CLS" token $\boldsymbol{x}_M^{(\text{Class})}$ for linear probing. Let $\boldsymbol{H}_{\text{all}} \in \mathbb{R}^{MDT+D}$ be the concatenated features and $\boldsymbol{W}_{\text{all}} \in \mathbb{R}^{(MDT+D)\times C}$ be the weights of the linear classifier, where C is the number of classes. After we learn the additional query tokens in VQT, we can freeze them and optionally employ group lasso to reduce the dimensionality of $\boldsymbol{H}_{\text{all}}$. Specifically, we follow HEAD2TOE [15] to first train the linear classification head with the group-lasso regularization $|\boldsymbol{W}_{\text{all}}|_{2,1}$, which encourages the ℓ_2 norm of the rows of $\boldsymbol{W}_{\text{all}}$ to be sparse. Then, the importance score of the i-th feature in $\boldsymbol{H}_{\text{all}}$ is computed as the ℓ_2 norm of the i-th row of $\boldsymbol{W}_{\text{all}}$. Finally, we select a fraction F of the features $\boldsymbol{H}_{\text{all}}$ with the largest importance scores and train a new linear head with the selected features.

B.3. Parameter Efficiency in Section 4.2

We provide the number of parameters for the transfer learning methods compared in subsection 4.2 of the main paper.

Linear-probing and full fine-tuning. For each task, linear probing only trains the prediction head and keeps the whole pre-trained backbone unchanged. Therefore, we only need to maintain one copy of the backbone that can be shared across all the downstream tasks. Contrastingly, full fine-tuning updates the whole network, including the backbone and the head, for each task. After training, each task needs to individually store its own fine-tuned backbone, thereby requiring more parameters.

VQT vs. HEAD2TOE. In subsection 4.2 of the main paper, We compare VQT with HEAD2TOE by matching the number of tunable parameters for each task in VTAB-1k. We provide details for this setup. More comparisons of VQT and HEAD2TOE can be found in subsection C.1.

HEAD2TOE [15] takes intermediate features from multiple distinct steps inside the pre-trained ViT backbone. For the features from each step, HEAD2TOE chooses a window size and a stride to perform average pooling to reduce the dimensionality. After concatenating the pooled intermediate features, HEAD2TOE further decides the fraction F for feature selection. In HEAD2TOE, the pooling window size, pooling stride and F are hyper-parameters picked based on validation accuracy.

For fair comparisons of VQT and HEAD2TOE [15], we match their numbers of tunable parameters used in different tasks. As both VQT and HEAD2TOE leverage intermediate features while keeping the backbone unchanged, the tunable parameters mainly reside in the final linear head. We focus on matching the feature dimension input to the classifier. First, we divide the 19 tasks in VTAB-1k into three groups; each group corresponds to a pair of pooling window sizes and strides that HEAD2TOE uses to generate the features *before* feature selection. Specifically, in the three groups, HEAD2TOE generates 68K-, 815K-, and 1.8M-dimensional features, respectively. Next, for simplicity, we set $F = 0.1$, which is the maximal F in HEAD2TOE's hyper-parameter search grid, for HEAD2TOE in all the 19 tasks. These results are obtained using the official HEAD2TOE released code[7]. For VQT, we choose T and F to match the final feature dimensions (after feature selection) used in HEAD2TOE. Specifically, we use $T = 1$ and $F = 0.7$, $T = 10$ and $F = 1.0$, and $T = 20$ and $F = 1.0$ for the three task groups, respectively.

Comparison on the number of parameters. We summarize the number of parameters needed for all the 19 VTAB-1k tasks in Table 4. As linear-probing shares the backbone among tasks and only adds linear heads that take the final "CLS" token, it only requires $1.01\times$ of the ViT-B/16 backbone parameters. By contrast, fine-tuning consumes

[7] https://github.com/google-research/head2toe

Methods	Total # of parameters
Scratch	19.01×
Linear-probing	1.01×
Fine-tuning	19.01×
HEAD2TOE	1.20×
VQT (Our)	1.22×

Table 4. Total numbers of parameters needed for all the 19 tasks, for the methods compared in Table 1. Each number represents how many times of one ViT-B/16 backbone's parameters (86M) are needed.

19.01× of the ViT-B/16 because each task maintains its own fine-tuned backbone. Compared to linear probing, VQT and HEAD2TOE use larger feature dimensions for prediction, thereby increasing the number of parameters in the final linear heads. Even so, VQT and HEAD2TOE are still parameter-efficient and need only 1.22× and 1.20× of the backbone parameters to learn 19 tasks.

B.4. Additional Training Details

We provide training details for VQT, VPT [23] and AdaptFormer [10] used in section 4.

For each task in VTAB-1k, we perform an 80/20 split on the 1K training images to form a training/validation set for hyper-parameter searching. After we pick the best hyper-parameters that yield the best validation accuracy, we use them for training on the original 1K training images. Finally, we report the accuracy on the testing set.

For training VQT with the ImageNet-1K backbone, we set T to match the number of tunable parameters with HEAD2TOE as described in subsection B.2. For training VQT with the MAE backbone (results reported in Table 3), we simply set $T = 1$ for all tasks in VTAB-1k. We then perform a hyper-parameter search to select the best learning rate from $\{1.0, 0.5, 0.25, 0.1, 0.05\}$ and the best weight decay from $\{0.01, 0.001, 0.0001, 0.0\}$. We use the Adam optimizer to train VQT for 100 epochs, and the learning rate follows the cosine schedule.

For training VPT with the ImageNet-21K and the MAE backbones, we use each task's best number of prompt tokens, which are released in VPT's GitHub page[8]. Since the VPT paper does not use the CLIP backbone, we simply set the number of tokens to 1 for all tasks in this case. We conduct a hyper-parameter search to pick the best learning rate from $\{1.0, 0.5, 0.25, 0.1, 0.05\}$ and the best weight decay from $\{0.01, 0.001, 0.0001, 0.0\}$. We train VPT using the Adam optimizer for 100 epochs with the cosine learning rate schedule.

Finally, when training AdaptFormer on all backbones, we use the bottleneck dimension $\hat{d} = 64$ and the scaling factor $s = 0.1$ following [10]. We similarly search the best learning rate from $\{1.0, 0.5, 0.25, 0.1, 0.05\}$ and the best weight decay from $\{0.01, 0.001, 0.0001, 0.0\}$ using the validation set. The AdaptFormer is trained with the Adam optimizer for 100 epochs, and the learning rate decay follows the cosine schedule.

[8]https://github.com/KMnP/vpt

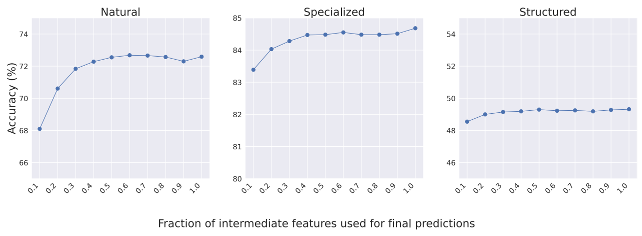

Figure 8. Average accuracy on VTAB-1k using different fractions of intermediate features for VQT.

C. Additional Experiments and Analyses

C.1. More Comparison with HEAD2TOE

In Table 1 of the main paper, we have compared VQT with HEAD2TOE under the constraint of using a similar number of tunable parameters, as mentioned in subsection B.3. To further evaluate the limit of VQT, we drop this constraint and allow both VQT and HEAD2TOE to select the best feature dimensions based on accuracy. Specifically, we pick the best feature fraction F for VQT using the validation set and compare it with the *best* HEAD2TOE results, which are also obtained by selecting the best feature dimension via hyper-parameter tuning, reported in their paper [15]. Table 5 shows the results of HEAD2TOE and VQT without the parameter constraint. VQT still significantly outperforms HEAD2TOE on 15 out of 19 tasks across the Natural, Specialized and Structured categories of VTAB-1k, demonstrating the effectiveness of the summarized intermediate features in VQT. We also compare HEAD2TOE and VQT on different pre-trained setups. As shown in Table 6, VQT consistently outperforms HEAD2TOE on supervised, self-supervised (MAE) and image-language (CLIP) pre-trained backbones. We used the best hyper-parameters from the HEAD2TOE paper for the ImageNet-1K backbone, and we only performed the learning rate and weight decay hyper-parameters search for the MAE and CLIP model. We match the number of tunable parameters in VQT and HEAD2TOE for fair comparisons.

C.2. Robustness to Different Feature Fractions

We study the robustness of VQT using different fractions of the intermediate features for prediction. Given a fraction F, we follow the strategy mentioned in subsection B.2 to select the best features. As shown in Figure 8, VQT is able to maintain its accuracy, with less than 1% drop, even when

Method	Natural								Specialized					Structured									
	CIFAR-100	Caltech101	DTD	Flowers102	Pets	SVHN	Sun397	Mean	Camelyon	EuroSAT	Resisc45	Retinopathy	Mean	Clevr-Count	Clevr-Dist	DMLab	KITTI-Dist	dSpr-Loc	dSpr-Ori	sNORB-Azim	sNORB-Elev	Mean	Overall Mean
HEAD2TOE	58.2	87.3	64.5	85.9	85.4	82.9	35.1	71.3	81.2	95.0	79.9	74.1	82.6	49.3	**58.4**	41.6	64.4	53.3	32.9	**33.5**	**39.4**	46.6	63.3
VQT (Ours)	**58.5**	**89.5**	**66.7**	**89.9**	**88.8**	**79.7**	35.1	**72.6**	**82.4**	**96.2**	**84.4**	**74.8**	**84.5**	**50.5**	57.1	**42.7**	**77.9**	**69.2**	**43.6**	24.1	32.0	**49.6**	**65.4**

Table 5. HEAD2TOE and VQT's test accuracies on the VTAB-1k benchmark with ViT-B/16 pre-trained on ImageNet-1K. In this comparison, we do not set parameter constraints and use the validation set to choose the best feature dimension based on accuracy. "Mean" denotes the average accuracy for each category and "Overall Mean" shows the average accuracy over 19 tasks.

Methods	Natural	Specialized	Structured	Mean
ImageNet-1K				
H2T	68.9	82.9	46.3	62.3
VQT	**72.7**	**84.5**	**49.3**	**65.3**
MAE				
H2T	55.6	80.3	44.4	55.7
VQT	**66.0**	**82.9**	**53.5**	**63.9**
CLIP				
H2T	69.3	82.0	33.8	57.0
VQT	**77.7**	**83.7**	**51.3**	**67.9**

Table 6. Performance comparison between HEAD2TOE (H2T) and VQT on various backbones.

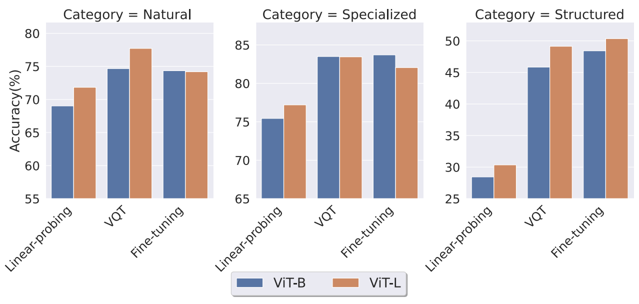

Figure 9. Performance comparison between *linear-probing*, *fine-tuning* and VQT on ViT-**B**ase (86M parameters) and ViT-**L**arge (307M parameters) pretrained on ImageNet-21K

we discard 60% of the features. On the Structured category in VTAB-1k, we can even drop up to 90% of the features for VQT without largely degrading its performance. These results reveal the potential of further compressing VQT to reduce more parameters.

C.3. Different Vision Transformer Backbones

Figure 9 shows the performance comparison between *linear-probing*, *fine-tuning* and VQT on ViT-**B**ase (86M parameters) and ViT-**L**arge (307M parameters) pretrained on ImageNet-21K. Generally speaking, all methods perform better on ViT-L than ViT-B due to higher model complexity. In the Natural and Specialized category, VQT has similar performance as *fine-tuning* on ViT-B and outperforms *fine-tuning* on ViT-L. As explained in subsection 4.2, the Natural and Specialized categories have stronger domain affinities with the source domain (ImageNet). Thus, both pre-trained backbones can generate more relevant intermediate features for similar domains. In the Structured category, *fine-tuning* slightly surpasses VQT on both backbones due to the difference between the pretrained dataset and the Structured category.

C.4. Variants of VQT

We ablate different design choices on the ViT-B pre-trained on ImageNet-21K and evaluate them on the VTAB dataset.

Summarized feature aggregation within layers. VQT relies on each layer's summarized features (the outputs of query tokens) for predictions. Although adding a suitable number of tokens can improve the performance as shown in Figure 6b, we investigate if we can effectively aggregate the summarized features within a Transformer layer to reduce the dimensionality by two approaches: (1) average pooling and (2) weighted-sum, as shown in Figure 12a. Specifically, (1) we perform pooling to average T output tokens back to *1* token; (2) we learn a set of weights for each layer to perform weighted-sum over T output tokens. After the aggregation step, the size of the summarized features for each layer will be changed from $\mathbb{R}^{D\times T}$ to $\mathbb{R}^{D\times 1}$.

Figure 11a and Figure 11b show the aggregation performance for T=10 and T=20 respectively. When we use T=10, average pooling performs similarly to T=10 and outperforms T=1 and weighted-sum. However, the trend is reversed when we use T=20; weighted-sum surpasses average pooling and T=1. To strike a balance between performance and efficiency, we suggest utilizing the validation set to choose a good within-layer aggregation method for a downstream dataset.

Summarized feature aggregation across layers. This section explores how to aggregate the summarized features (the outputs of query tokens) across layers. Instead of concatenation (*Concat*), the default method we use in the main

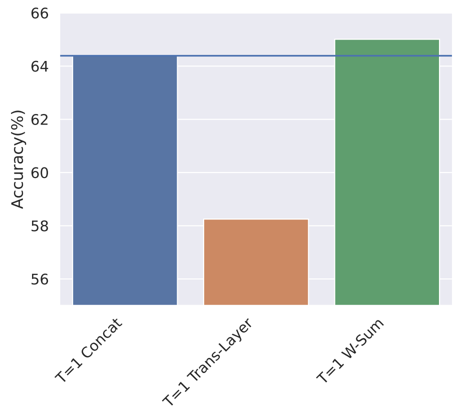

Figure 10. Performance comparison for different across-layer aggregation methods when T=1. The blue line shows the accuracy for T=1 *Concat*. *W-Sum* is a more efficient and effective way to aggregate summarized features across layers since it reduces the dimensionality and performs better.

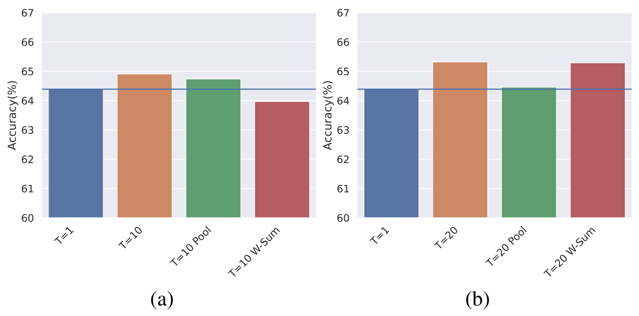

Figure 11. Performance comparison for different within-layer aggregation methods when T=10 and T=20, where "pool" and "W-Sum" refers to average pooling and weighted sum, respectively. The blue line shows the accuracy for T=1. Note that the summarized feature dimension for T=10 (20) pool (w-Sum) is the same as the one for T=1.

paper, we try feeding the summarized features from all layers to a randomly initialized Transformer layer with the CLS token from the last Transformer layer and use the output of the CLS token for prediction, dubbed *Trans-Layer*. We also try to perform weighted-sum over all the summarized features, dubbed *W-Sum*. When T=1, the dimension for Concat is $\mathbb{R}^{D\times M}$ where M is the number of Transformer layers in the backbone and the dimension for *Trans-Layer* and *W-Sum* is $\mathbb{R}^{D\times 1}$. The across-layer aggregation methods are demonstrated in Figure 12b.

As shown in Figure 10, *Trans-Layer* is way behind *Concat*. We hypothesize that the limited number of images per task may not be sufficient to train a randomly initialized Transformer layer. On the contrary, *W-Sum* outperforms the default *Concat*, which is surprising for us since the same dimension of the summarized feature in different layers may represent different information, and thus, the summarized

Methods	Natural	Specialized	Structured	Mean
VPT	74.9	82.9	53.9	65.9
VPT+H2T	69.1	81.1	50.9	64.0
VPT+VQT	**76.8** (6/7)	**83.8**(2/4)	**53.4**(6/8)	**68.4**
AF	73.4	80.1	47.3	63.8
AF+H2T	69.4	82.3	51.4	64.5
AF+VQT	**77.0**(7/7)	**84.6**(2/4)	**53.4**(6/8)	**68.7**

Table 7. Compatibility comparison between HEAD2TOE (H2T) and VQT on VPT and AdaptFormer (AF). The (/) represents the number of wins compared to baselines and baselines+H2T. The results are based on ImageNet-1k pre-trained backbone.

feature from different layers may not be addable. However, based on this result, we hypothesize that the skip connection in each layer can be the cause of the addibility of summarized features from different layers. We believe studying more effective and efficient aggregation methods for the summarized features is an interesting future direction.

C.5. t-SNE Visualization for More Datasets

We present t-SNE visualizations of the CLS token and our summarized features for more tasks in Figure 13. Similar to Figure 5, adding summarized features makes the whole features more separable than the CLS token alone, demonstrating the benefit of using intermediate features and the advantage of our query tokens in summarizing them.

C.6. Results for All Tasks on Different Backbones

Table 8 shows the per-task accuracies for 19 tasks in VTAB on different ViT-B backbones, including CLIP, MAE and Supervised ImageNet-21K.

C.7. Compatibility comparison between VQT and H2T

We compare the compatibility performance between HEAD2TOE and VQT with VPT and AdaptFormer (AF). For a fair comparison, we ensure that the output feature dimension is the same as the original one (D=768 in ViT) when we combine VPT and AdaptFormer with HEAD2TOE and VQT. We use the default feature selection method in the original paper for HEAD2TOE and the weighted-sum approach (see subsection C.4 for details) for VQT. Table 7 shows the results on ImageNet-1k pre-trained backbone and VQT demonstrates more competitive compatibility performance than HEAD2TOE.

D. Additional Discussions

D.1. More Discussions on Memory Usage

As mentioned in the last paragraph of subsection 3.3 and as shown in subsection 4.4, since VQT keeps all the intermediate features intact and only learns to tune the query

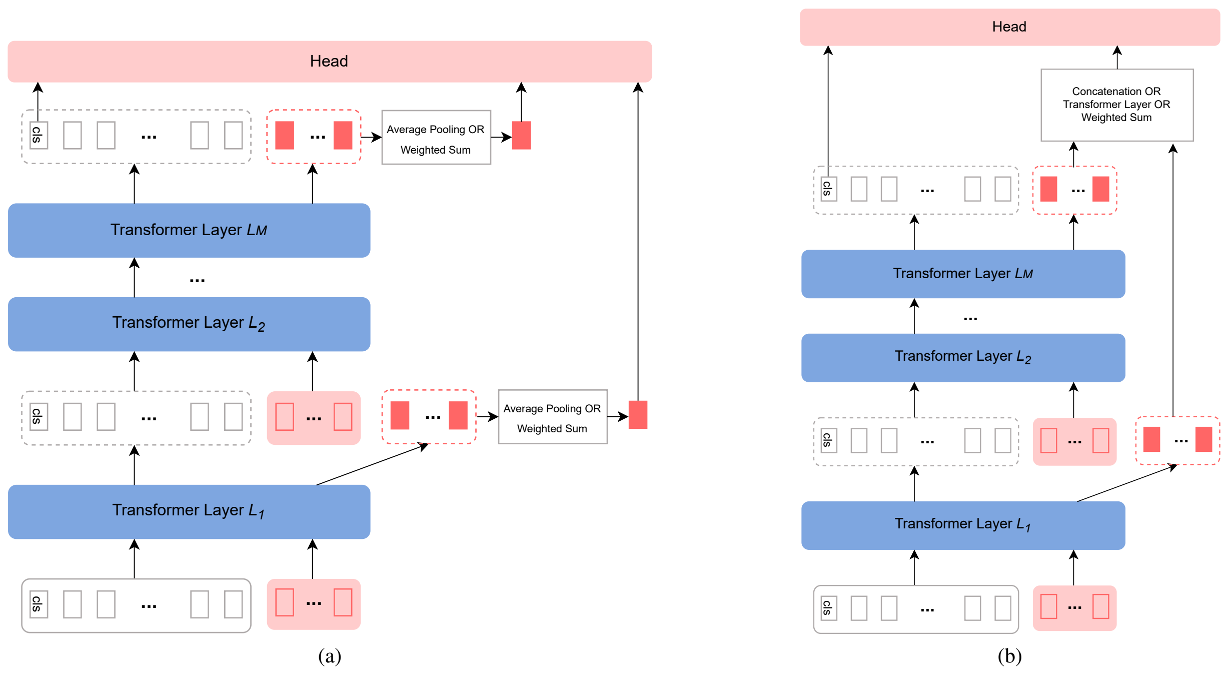

Figure 12. (a) shows the within-layer aggregate methods. Multiple output query tokens within the same layer can be aggregated through average pooling or weighted sum. (b) shows the across-layer aggregation methods. Output query tokens from different layers can be aggregated through concatenation, weighted sum or another randomly initialized Transformer layer.

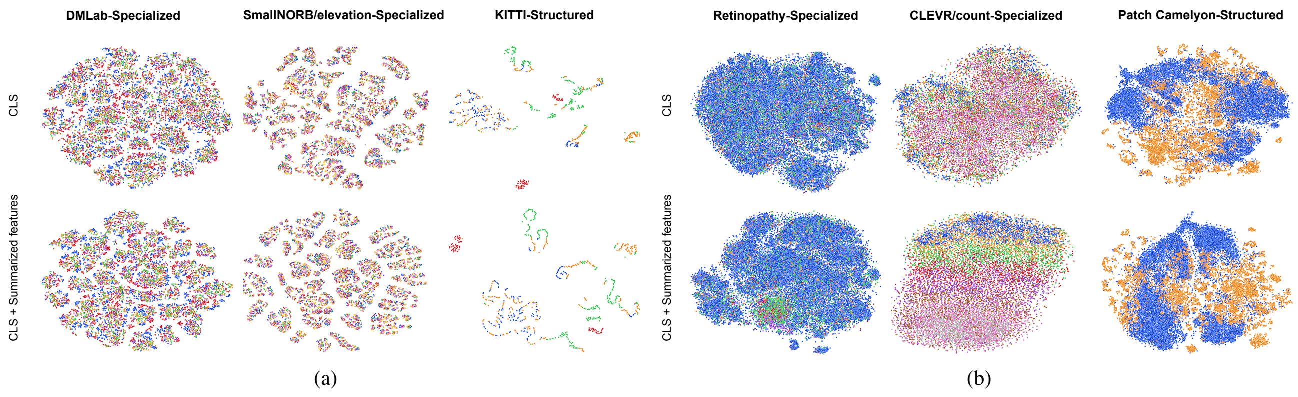

Figure 13. **t-SNE visualization of the CLS tokens alone (top) and CLS tokens plus our summarized features (bottom)** on more tasks from VTAB. Adding the summarized intermediate features makes the whole features more separable. We include tasks that have less or equal to 10 classes for visualization.

tokens and the prediction head, the training process bypasses the expensive back-propagation steps and does not require storing any intermediate gradient results, making it very memory-efficient. As shown in Figure 1a, VPT needs to run back-propagation (red arrow lines) through the huge backbone in order to update the inserted prompts. On the contrary, VQT only needs gradients for the query tokens because all intermediate output features are unchanged, as shown in Figure 1b.

D.2. Cost of VQT and AdaptFormer

In subsection 4.3, to confirm that the improvement mentioned above does not simply come from the increase of tunable parameters, we enlarge AdaptFormer's added modules by increasing the bottleneck dimension $\hat{d}$ from 64 to 128 and 256 to match the tunable parameter number of AdaptFormer when equipped with VQT. Here, we show the detailed parameter calculation in Figure 2. The additional parameters for AdaptFormer and VQT can be calculated

	Natural								Specialized					Structured									
Method	CIFAR-100	Caltech101	DTD	Flowers102	Pets	SVHN	Sun397	Mean	Camelyon	EuroSAT	Resisc45	Retinopathy	Mean	Clevr-Count	Clevr-Dist	DMLab	KITTI-Dist	dSpr-Loc	dSpr-Ori	sNORB-Azim	sNORB-Elev	Mean	Overall Mean
CLIP backbone																							
AdaptFormer	73.7	93.2	75.2	96.8	90.7	92.7	56.1	82.6	83.3	95.7	87.8	73.6	85.1	76.5	61.9	49.6	84.1	84.6	55.4	29.5	45.7	60.9	74.0
AdaptFormer+VQT	71.3	95.3	77.1	96.2	90.6	93.3	51.2	82.1	84.8	96.4	88.7	73.4	85.8	75.8	62.6	52.4	83.8	91.8	54.6	33.6	46.5	62.6	74.7
VPT	66.3	90.1	73.7	94.7	90.3	91.6	56.0	80.4	83.3	93.4	87.3	75.6	84.9	41.5	57.5	52.3	80.7	65.1	54.3	27.7	28.4	50.9	68.9
VPT+VQT	70.8	95.1	72.7	93.8	89.8	93.5	54.8	81.5	85.2	95.7	89.7	74.8	86.3	52.5	62.6	55.3	84.1	77.1	56.4	34.6	35.1	57.2	72.3
MAE backbone																							
AdaptFormer	53.5	90.1	60.3	83.3	81.4	83.0	29.6	68.7	83.0	93.9	74.4	73.8	81.3	77.8	60.3	44.0	79.5	75.9	53.1	30.3	45.6	58.3	67.0
AdaptFormer+VQT	56.8	90.4	63.7	86.8	80.7	89.7	29.7	71.1	84.5	95.4	80.9	72.5	83.3	65.9	58.5	46.5	84.0	82.2	53.2	32.1	51.1	59.2	68.6
VPT	45.5	88.9	62.2	75.1	73.2	75.2	24.4	63.5	80.1	94.6	68.3	73.6	79.1	69.5	58.2	39.4	70.8	53.6	51.2	20.4	25.5	48.6	60.5
VPT+VQT	48.9	90.3	65.2	87.4	81.8	75.9	26.0	67.9	81.4	95.1	80.8	73.6	82.7	63.3	59.2	44.4	80.2	46.5	52.7	22.8	28.4	49.7	63.4
Supervised ImageNet-21K backbone																							
AdaptFormer	79.9	89.8	68.5	98.0	88.3	81.4	54.8	80.1	80.3	95.4	81.1	72.3	82.3	71.0	55.0	42.3	68.8	65.9	45.1	24.9	29.8	50.3	68.0
AdaptFormer+VQT	77.1	93.7	68.2	98.2	89.8	84.1	45.9	79.6	82.1	96.2	85.6	73.2	84.3	71.4	54.9	44.5	72.3	76.7	45.2	27.6	31.3	53.0	69.4
VPT	79.8	89.9	67.5	98.0	87.0	79.4	52.3	79.1	83.5	96.0	83.7	75.2	84.6	68.1	60.1	43.0	74.8	74.4	44.4	30.0	40.2	54.4	69.9
VPT+VQT	76.8	92.6	69.2	98.3	87.8	81.6	46.2	78.9	81.3	96.3	84.7	72.4	83.7	59.6	60.3	43.0	77.6	79.3	46.0	31.2	39.5	54.6	69.7

Table 8. Test accuracies for AdaptFormer, VPT and their combinations with VQT on the VTAB-1k benchmark on ViT-B/16 pre-trained with CLIP, MAE and Supervised ImageNet-21K. "Mean" denotes the average accuracy for each category and "Overall Mean" shows the average accuracy over 19 tasks.

as $\hat{d} \times 2 \times D \times M$ and $\underbrace{T \times D \times M}_{\text{query tokens}} + \underbrace{T \times D \times M \times C}_{\text{prediction head}}$, respectively where $\hat{d}$ denotes the bottleneck dimension of AdaptFormer; D is the embedding dimension; M is the number of Transformer layer; T represents the number of VQT's query tokens; C denotes the average number of classes in VTAB, and we round it to 50 for simplicity. The numbers of tunable parameters and percentages of tunable parameters over ViT-B's number of parameters (86M) for AdaptFormer and AdaptFormer+VQT are shown in Table 9.

D.3. Training Efficiency

In this subsection, we point out another *potential* advantage of VQT besides its parameter and memory efficiency — training efficiency (*i.e.*, the number of floating-point operations and the overall wall-clock time in training). This can be seen from two aspects.

On the one hand, since VQT does not change the original intermediate features obtained from the pre-trained backbone but only learns to combine them, we can pre-compute them for all the downstream data and store them in the hard drive or even random access memory (RAM)[9] for later training (epochs). As mentioned in subsection B.4, we perform 100 epochs for learning the query tokens in VQT, in which we indeed only need to compute the intermediate features once in the first epoch, and reuse them in later epochs. Given a standard ViT-B with 12 Transformer layers and 197 embedding tokens of size 768 for each layer, all the intermediate features for an image amount to "$12 \times 197 \times 768$" 32-bit floats (7MB); storing them for a task in VTAB with 1K images only requires 7GB in the hard drive or RAM. With all the pre-computed intermediate features, we can parallelize the forward and backward passes of the 12 layers at the same time, potentially making the training process 12× faster.

On the other hand, since VQT only uses the outputs of the newly introduced query tokens for predictions, during the forward pass within each layer, we just need to pass the MSA features corresponding to these tokens to the MLP block, making it additionally faster on top of the cross-layer parallelization mentioned above.

[9]We note that these are not the same memory as in memory efficiency. The latter refers to the GPU or CPU memory.

AdaptFormer	$\hat{d} = 64$	$\hat{d} = 128$	$\hat{d} = 256$
Tunable parameters #	1179648	2359296	4718592
Tunable parameters %	1.37%	2.74%	5.49%
AdaptFormer+VQT	$\hat{d} = 64$ &	$\hat{d} = 64$ & $T = 2$	$\hat{d} = 64$ & $T = 4$
Tunable parameters #	1179648	2119680	3059712
Tunable parameters %	1.37%	2.46%	3.56%

Table 9. Numbers of tunable parameters and percentages of tunable parameters over ViT-B's number of parameters (86M) for AdaptFormer with different bottleneck dimensions and AdaptFormer+VQT with different numbers of query tokens.