A APPENDIX

A.1 RELATED WORK

A.1.1 DYNAMIC HALTING ALGORITHM

Adaptive Computation Time (ACT) algorithm (Graves, 2016) was first proposed for dynamic halting in recurrent neural networks. Dehghani et al. (2018) adapted this algorithm to transformer-based models and proposed universal transformer. Universal transformer with shared trainable parameters across transformer blocks has adaptive depth for different samples, and this work shows that the adaptivity enables the transformer to solve some tasks that the vanilla transformer totally failed. Yin et al. (2022) adapted ACT algorithm to the vision transformer along the depth. However, the ACT-based transformer training is not as stable as the vanilla transformer and it needs careful hyper-parameters tuning. To overcome this weakness, PonderNet (Banino et al., 2021) improves the universal transformer by reformulating the halting policy as a probabilistic model. Balagansky & Gavrilov (2022) further stabilizes this training process by removing the sampling process in PonderNet. Another way to halt adaptively is confidence-based dynamic halting algorithms. Schuster et al. (2021; 2022); Schwartz et al. (2020) investigated the confidence-based dynamic halting algorithms on adaptive model depth.

Our work also proposed a dynamic halting algorithm, but we are interested in pondering at input sequence length instead of model depth. Another difference is that we have no trainable layers to compute the halting score, which is a requirement of adaptive transformer input. Alternatively, we use the entropy of logits from the token selection layer as an indicator to make a halting decision.

A.1.2 ADAPTIVE SEQUENCE LENGTH

Adaptive sequence length can be achieved by token pruning. For instance, Meng et al. (2022) prune the patch tokens in the vision transformer via a light-weight decision network, and Fayyaz et al. (2022) drop the patch tokens by sampling adaptively. Our AdaTape is different from their work as AdaTape appends a small number of tokens to the original input adaptively instead of pruning the intermediate token representations within the model. Also, the content of the tokens in the existing work is not adaptive. The appended tokens in AdaTape are sparsely selected from the tape bank, which is helpful to improve the effectiveness of the transformer.

Wang et al. (2021) proposed another way to achieve adaptive sequence length for ViT. The proposed model uses a large patch size at first and decreases the patch size adaptively for different samples. This is similar to our input-driven bank. However, first, in our input-driven bank, instead of using all smaller patches, we use only a subset of the fine-grained patches adaptively. In addition, instead of limiting the setup to image-only inputs, AdaTape supports a learnable bank and formulates this problem in a more general way.

A.1.3 EXTRA TOKENS

Extra tokens are another important component of AdaTape. Song et al. (2021) use an extra special token for vision transformer-based object detection. Burtsev et al. (2020) introduces memory tokens to augment transformer capacity. Using learnable prompt tokens in NLP (Lester et al., 2021) can be also perceived as adding extra tokens to the input. In all these works, the number of added tokens is fixed across different samples and often uses a deterministic set of tokens (imagine having more than one `[CLS]` token). However, AdaTape selects and appends a variable number of tokens from the bank adaptively per sample. Such an adaptive input sequence provides not only a larger capacity but also a flexible computation budget.

A.2 INPUT-DRIVEN BANK DETAILS

We introduce more details about the input-driven bank of AdaTape. We generate the input-driven bank by:

$$\boldsymbol{Z}_{bank} = h_2(h_1(\boldsymbol{X}_p) + \boldsymbol{E}_{pos}) \tag{1}$$

where $\boldsymbol{X}_p$ is the input sequence with fine-grained tokenization (*e.g.,* smaller patch size for ViT), h_1 and h_2 are both trainable linear projection, $\boldsymbol{E}_{pos}$ is position embedding. Since $\boldsymbol{Z}_{bank}$ is conditioned on the input sample, we need to generate the content of the bank on-the-fly. This is also the reason why an input-driven bank is slightly more computationally expensive than a learnable bank.

A.3 Scaling with Patch Size

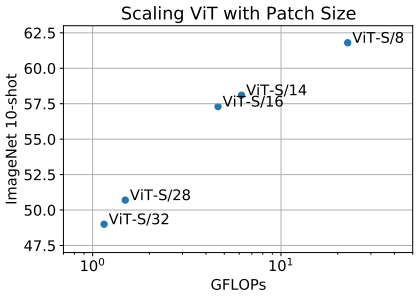

Figure 5: We scale ViT with patch size. The model is pre-trained on JFT-300M (Sun et al., 2017). We report the few-shot performance on ImageNet (Deng et al., 2009)

We fix all other parameters and scale ViT with a smaller patch size to check the effectiveness of smaller patches. We experiment with ViT-S which will not become expensive as the sequence length grows. For example, training ViT-B/8 with data parallelism on 512 TPUv3 cores has the OOM issue. In Figure 5, when adding more computation, although there is a significant accuracy improvement, but more patches introduce very expensive computation costs. Such a trade-off inspires us to use smaller patches as the source to generate bank for a efficient fine-grained data modeling.

A.4 LayerNorm is making ACT invaild in adaptive sequence

We argue that naively applying layer normalization in the transformer invalidates the ACT algorithm in AdaTape. To justify this, we first assume we are using a transformer with the following standard transformer architecture:

$$\boldsymbol{X}' = \mathrm{MSA}(\mathrm{LayerNorm}(\boldsymbol{X})) + \boldsymbol{X} \tag{2}$$

$$\boldsymbol{X}'' = \mathrm{FFN}(\mathrm{LayerNorm}(\boldsymbol{X}')) + \boldsymbol{X}' \tag{3}$$

where $\mathrm{MSA}(\cdot)$ is multi-head attention layer and $\mathrm{FFN}(\cdot)$ is feed-forward network. We are going to replace $\boldsymbol{X}$ by $p_t\boldsymbol{z}_t$, where p_t is the halting score of t^{th} tape token $\boldsymbol{z}_t$. Then, the output of first $\mathrm{LayerNorm}(\cdot)$ should be:

$$\mathrm{LayerNorm}(p_t\boldsymbol{z}_t) = \frac{p_t\boldsymbol{z}_t - \frac{1}{H}\sum_{h=1}^{H} p_t z_{t,h}}{\sqrt{\frac{1}{H}\sum_{h=1}^{H}(p_t z_{t,h} - \sum_{j=1}^{H} p_t z_{t,j})^2 + \epsilon}} * \gamma + \beta \tag{4}$$

since ϵ is a very small constant, it is reasonable to write the equation above as:

$$\mathrm{LayerNorm}(p_t\boldsymbol{z}_t) \approx \frac{\boldsymbol{z}_t - \frac{1}{H}\sum_{h=1}^{H} z_{t,h}}{\sqrt{\frac{1}{H}\sum_{h=1}^{H}(z_{t,h} - \sum_{j=1}^{H} z_{t,j})^2 + \epsilon}} * \gamma + \beta = \mathrm{LayerNorm}(\boldsymbol{z}_t) \tag{5}$$

We can see p_t is ignored during the normalization, which means the value of p_t cannot change the output of the first $\mathrm{LayerNorm}(\cdot)$ in practice. Since p_t is the output of the trainable layer $g(\cdot)$ in ACT, the $g(\cdot)$ cannot be really trained well by back-propagation. To alleviate this issue, one possible solution is applying the weight p_t to tape tokens after normalization layers. This is a simple and straightforward solution that may make sense. However, empirically, we observe the weights of all tokens will increase to 1.0 very fast in practice and the model will then always only append one token even if we do not use any loss function to penalize longer sequence. We suggest the reason is

that uniformed token mean and variance are highly desired by attention layers. Anyhow, this result is not expected because there is no adaptive ability observed. In addition, we observed the training is also extremely unstable under this design.

In summary, we cannot apply scalar weight to the tape token, so we cannot have a trainable linear layer to compute the halting score p. That makes the ACT-based dynamic halting mechanism, including PonderNet, not applicable to AdaTape. We, therefore, are required to design a new adaptive computation algorithm for elastic input sequence, *i.e.,* Adaptive Tape Reading. Such a reasoning process can also be extended to other future conditional and adaptive computation work. For instance, if you want to feed the layer norm with one token after applying weight on the token, you need to be careful about where is the weight from. If this weight is from a trainable layer like ACT, we must check whether this layer can be well-trained via reasoning.

A.5 Configuration for Transformer-Tiny

Table 3: Transformer configuration for all tiny-level models.

	Transformer-Tiny
Depth	12
Hidden Dimension	196
MLP Dimension	768
#Attention Heads	3

We use the tiny configuration for all models on the parity task. As shown in Table 3, the Tiny level transformer still uses 12 layers, which is the same as the transformer base. The difference is that tiny configuration has smaller hidden dimensions and fewer attention heads.

A.6 Hyper-parameters

Table 4: Hyper-parameters for AdaTape on image classification

	AdaTape-Learn	AdaTape-Input
Max ponder times T	10	10
Halting threshold τ	2.0	2.0 (B) /1.0 (L)
Loss weight λ	0	0.01
Bank Size C	10000	784

On JFT-300M pre-training, we follow Dosovitskiy et al. (2020) and train all models for 7 epochs. We use the same learning rate, batch size, and learning schedule. Customized hyper-parameters for AdaTape are summarized in Table 4. We employed the fixed max ponder times for all models. Smaller τ on AdaTape-L with an input-driven bank. The bank size is 10000 for AdaTape-learn. We use bank size 784 for AdaTape Input as we set patch size as 8 to generate tokens from images with 224×224 resolution. AdaTape with a learnable bank can be trained without halting loss. Also, note that we append tape tokens after first transformer encoder layer for better query quality and tape selection.

For ImageNet training from scratch, we summarized the data augmentation and corresponding hyper-parameters in Table 5. Similar with existing work (Beyer et al., 2022), we used Mixup (Zhang et al., 2017), RandAug (Cubuk et al., 2020) and label smoothing (Szegedy et al., 2016) to improve the robustness.

A.7 Tricks of Learnable Bank Training

We found the training of AdaTape with a learnable bank is relatively unstable. To alleviate this, we propose two tricks to improve the training process. The core idea of these two tricks is to improve the diversity of tape tokens and encourage the model to explore the tape bank. We first add noise

Table 5: Hyper-parameters when training on ImageNet-1K only. Ti, S and B denote Tiny, Small, and Base scales.

Name	Value
Learning Rate	0.001
Linear Warmup Steps	10000
Learning Rate Decay	Cosine Decay
Optimizer	AdamW
(β_1, β_2)	$(0.9, 0.999)$
Weight Decay	1e-4(Ti), 8e-5(S&B)
Epoch	300
Batch Size	1024
Mixup	0.2(Ti), 0.5(S&B)
Label Smoothing	0(Ti), 0.1(S&B)
RandAug	(2,10)(Ti), (2,15)(S&B)

to query $\boldsymbol{q} = \boldsymbol{q} + \lambda\ \epsilon$, where ϵ is sampled from standard normal distribution and λ is the weight of noise. We set λ as 0.01 during training. We also mask a subset of the tape tokens in the bank randomly. To implement this, we initialize $\boldsymbol{m} = \mathbf{0} + \boldsymbol{b}$ for ATR algorithm, where $\boldsymbol{b} \in \mathbb{R}^{1\times C}$ and $b_c \sim \text{Bernoulli}(p)$. We set p as 0.1 by default. We also observed that larger p can further improve the training stability.

A.8 More results on ImageNet

Table 6: Results of training on ImageNet-1K only. We use the input-dirven bank for AdaTape. * denotes that we approximate the throughput by the models with similar architecture and input pipeline. For A-ViT, we not only report their results from the paper but also re-implement A-ViT by training from scratch, *i.e.,* A-ViT(Ours).

Model	Adaptive	#Param	Throughput	ImageNet Top-1 (%)
ViT-Ti/16		5.7	387.5	58.7
DeiT-Ti/16		5.7	381.8*	71.3
PlainViT-Ti/16		5.7	381.8	73.0
U2T-Ti/16	✓	6.1	364.3	70.2
A-ViT-Ti/16	✓	5.7	226.6*	71.0
A-ViT-Ti/16(Ours)	✓	5.7	226.6	73.2
AdaTape-Ti/16	✓	6.3	380.9	**73.6**
ViT-S/16		22.0	385.7	75.2
DeiT-S/16		22.0	365.8*	78.9
PlainViT-S/16		22.0	365.8	79.2
U2T-S/16	✓	23.8	163.2	74.5
A-ViT-S/16	✓	22.0	166.6*	78.6
A-ViT-S/16(Ours)	✓	22.0	166.6	77.0
AdaTape-S/16	✓	24.3	366.5	**79.5**
PlainViT-B/16		87.1	159.9	79.5
AdaTape-B/16	✓	94.9	130.5	**80.8**

We train with ImageNet-1K only and summarized the results in Table 6. We can see AdaTape outperforms all adaptive baselines by a large margin. Even compared to highly-optimized baselines without adaptivity like PlainViT (Beyer et al., 2022) and DeiT (Touvron et al., 2021), AdaTape can still surpass them with a comparable computation budget. For instance, AdaTape-S/16 uses only $0.4\times$ training cost and $0.3\times$ parameters but achieves almost comparable results with PlainViT-B/16. We may also note that Tiny scale models cannot achieve much higher throughput than Small scale models on ImageNet. We suggest the reason is that the efficiency bottlenecks are mainly from data loading and preprocessing instead of the computation budget within neural networks.

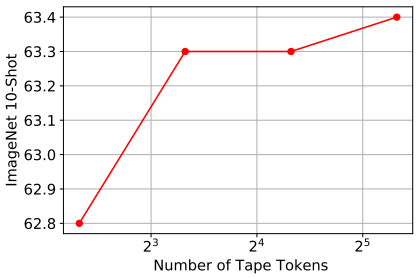

Figure 6: We sweep the number of tape tokens over {5, 10, 20, 40}. Considering more tape tokens mean much more computation cost, we set 10 as our default choice.

A.9 Ablation on Hyper-Parameters

Number of tape tokens We investigate the effect of the number of tape tokens in this section. In the model with adaptive sequence length, the number of tape tokens is controlled by the model adaptively. To control the real length directly, we use the AdaTape without adaptive length as a platform. We sweep the number of tape tokens T over {5, 10, 20, 40} and summarize the results in Figure 6. We can see an obvious improvement when we increase the T from 5 to 10. However, the model is saturated after that. Since using a longer sequence means more computation budget, we select to use 10 tape tokens as the default choice.

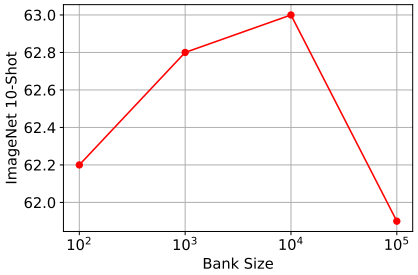

Figure 7: We sweep the bank size over {1e+2, 1e+3, 1e+4, 1e+5} for AdaTape with a learnable bank, and found 1e+4 is the sweep point.

Bank Size We also sweep the bank size over {1e+2, 1e+3, 1e+4, 1e+5} for AdaTape with a learnable bank. As shown in Figure 7, the AdaTape performs best when we have $1e + 4$ tape tokens in the bank. As we fix the number of recurrences as 10 in ATR algorithm by default, a larger bank will only increase the computation cost linearly.

A.10 Ablation on Computation Cost

Although AdaTape only increases the computation cost slightly, to further verify the improvement is from more reasonable design and adaptive abilities instead of more computation, we conduct ablation experiments on computation cost. First, as shown in Figure 8, even if a smaller patch size can improve ViT, AdaTape is still outperforming ViT with less computation. In addition, we report the quality-cost comparison in terms of throughput in Figure 9. We can see AdaTape can outperform ViT significantly with smaller latency.

A.11 Ablation on Number of Trainable Parameters

Since we have two FFNs in every transformer block, AdaTape has more trainable parameters than ViT. To validate that the improvement is not just caused by having more parameters, we increase the

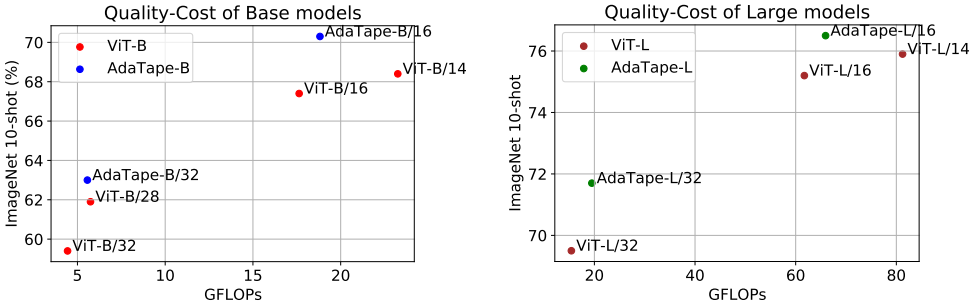

Figure 8: Ablation Study on the quality-cost trade-off. We scale the standard transformer (*i.e.,* ViT) to a smaller patch size, *e.g.,* ViT-B/14 and ViT-L/14, and compare it with AdaTape.

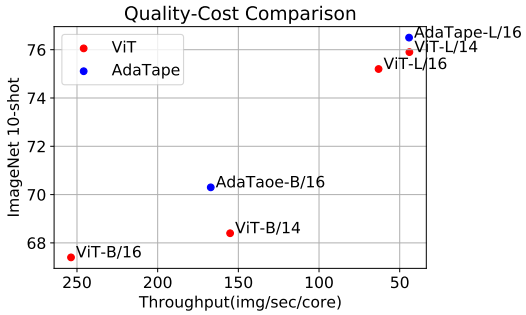

Figure 9: we report the quality-cost comparison in terms of throughput to verify that AdaTape is better than baselines with comparable computation cost.

trainable parameters in ViT by adding 2 more FFN layers to process different tokens. Then, ViT has 3 FFNs in total. The first FFN is used to handle `[CLS]` token. The second FFN and the third FFN are fed by half of the patch tokens, respectively. The results are summarized in Table 7. We can see AdaTape outperforms ViT with 3 FFNs using less computation and fewer parameters. For instance, AdaTape-B/16 surpasses ViT-B/14-3FFN by 1.3% in terms of top-1 accuracy on ImageNet 10-shot.

A.12 Visualization of Token Selection Distribution

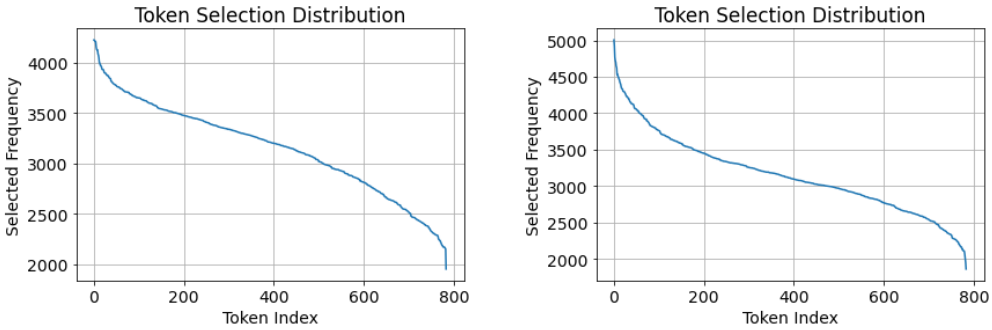

Figure 10: We visualize the tape token selection distribution in AdaTape-B/32 (left) and AdaTape-B/16 (right). The index id is sorted by the value of the selected frequency

Table 7: Ablation study on the number of trainable parameters.

Model	GFLOPs	Throughput	#Param	IN 10-shot
ViT-B/28	5.730	661.1	100.9	61.9
ViT-B/28-3FFN	5.744	517.0	200.1	62.4
AdaTape-B/32	5.585	431.8	185.6	63.0
ViT-B/14	23.254	155.1	99.7	68.4
ViT-B/14-3FFN	23.239	148.8	198.9	69.0
AdaTape-B/16	18.837	167.1	192.5	70.3

Similar to Section 3.5, we collect the token selection results on JFT-300M validation set. We sort the tape token index by the frequency and visualize the token selection distribution in Figure 10. We can observe the token selection decision obey long-tail distribution. That shows our AdaTape prefers the tape tokens at some specific positions, which is similar to our observation in Figure 4, *i.e.,* central patches are frequently selected.