ADAPTIVE COMPUTATION WITH ELASTIC INPUT SEQUENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

When solving a problem, human beings have the adaptive ability in terms of the type of information they use, the procedure they take, and the amount of time they spend approaching and solving the problem. However, most standard neural networks have the same function type and fixed computation budget on different samples regardless of their nature and difficulty. Adaptivity is a powerful paradigm as it not only imbues practitioners with flexibility pertaining to the downstream usage of these models but can also serve as a powerful inductive bias for solving certain challenging classes of problems (Dehghani et al., 2018; Banino et al., 2021). In this work, we propose a new strategy, AdaTape, that enables dynamic computation in neural networks via adaptive tape tokens. AdaTape employs an elastic input sequence by equipping an existing architecture with a dynamic read and write tape. Specifically, we adaptively generate input sequences using tape tokens obtained from a tape bank that can either be trainable or generated from input data. We analyze the challenges and requirements to obtain dynamic sequence content and length, and propose the Adaptive Tape Reader (ATR) algorithm to achieve both objectives. Via extensive experiments on image recognition tasks, we show that AdaTape can achieve better performance while maintaining the computational cost.

1 INTRODUCTION

Adaptive computation is central to human intelligence. This is clear, given that humans spend a variable amount of time and energy on different problems depending on their complexity (Meunier et al., 2009). Adaptivity in neural networks is attractive for two key reasons. Firstly, adaptive computation could potentially be a powerful and essential inductive bias for solving challenging problems that would have been significantly harder otherwise (Dehghani et al., 2018; Banino et al., 2021). Secondly, adaptive computation could imbue practitioners with downstream flexibility pertaining to the usage of these models. For the most part, altering the computation budget of a model after it has been trained becomes almost impossible. Hence, the ability to flexibly scale computational costs and budget dynamically is highly desirable.

This paper proposes AdaTape, a new general-purpose adaptive computation method. The key idea is to introduce elastic input sequences via the means of a dynamic read and write memory tape. Unlike all prior works that investigate adaptivity via sparse conditional computation (Fedus et al., 2022; 2021; Lepikhin et al., 2020) or adaptivity through recursion over architecture (Dehghani et al., 2018; Banino et al., 2021; Graves, 2016), this work presents a new perspective that explores adaptivity with respect to input sequence length (or read/write memory tapes from the perspective of a Neural Turing Machine (Graves et al., 2014)). We postulate that this exploration is crucial for the development of this class of methods and is very complementary to the existing suite of methods developed to encourage adaptive computation in neural networks.

AdaTape promotes adaptivity in both qualitative and quantitative aspects. Specifically, AdaTape controls (1) the contents of the memory tape and (2) how much memory tape is used. To this end, AdaTape is characterized by a readable tape bank that can be dynamically read from using a newly proposed dynamic halting algorithm which we call Adaptive Tape Reader (ATR). Concretely, this newly proposed ATR method adaptively and dynamically selects the content and length of this memory tape which is appended to the inputs of a standard Transformer (Vaswani et al., 2017).

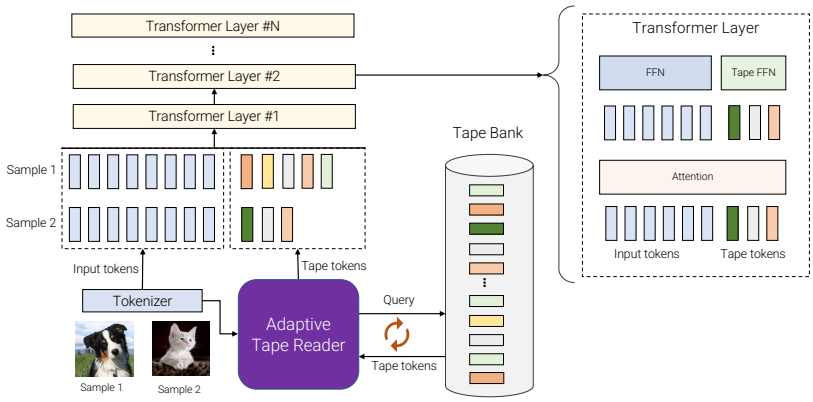

Figure 1: An overview of AdaTape. For different samples, we pick a variable number of different tokens from the tape bank. The tape bank can be driven from input, e.g., by extracting some extra fine-grained information or it can be a set of trainable vectors. The Adaptive Tape Reader is used to recursively select different sequences of tape tokens, with variable length, for different inputs. These token are then simply appended to inputs and fed to the transformer encoder.

Given that increasing computation budget generally leads to improved quality (Kaplan et al., 2020; Dehghani et al., 2021; Hoffmann et al., 2022; Zhai et al., 2022; Abnar et al., 2021), this enables a new way for adaptively scaling the computation budget without adding new parameters or applying a part of the model recursively.

To ascertain the effectiveness of the proposed AdaTape method, we first evaluate it on the challenging Parity task (Graves, 2016; Banino et al., 2021), a standard verification check for Adaptive Computation Time (ACT) algorithms (Graves, 2016). Our results demonstrate that AdaTape performs well on this problem. Meanwhile, this problem remains completely unsolvable by vanilla Transformers. This not only verifies that the AdaTape inductive bias is crucial in solving certain classes of problems but also asserts its correctness. Given that the standard Transformer is touted as the true universal algorithm with ubiquitous impact across all fields (Jumper et al., 2021; Dosovitskiy et al., 2020; Vaswani et al., 2017), it would be unthinkable if Transformers lack the inductive bias for a standard vector parity problem.

Finally, we conduct large-scale experiments on vision tasks (e.g., image recognition and evaluate their few-shot accuracy), showing that AdaTape performs well and outperforms vanilla Transformers when *compute-matched* (Dehghani et al., 2021) across both FLOPS and throughput. While AdaTape does not improve efficiency during training, the granted flexibility and adaptivity allow dynamic scaling of the computation budget during inference. Given that the standard practice is to train and serve n models to potentially cater to variable computation budgets (e.g., base model for less important workloads and large models for prioritized examples), we consider the property of having a single model flex between multiple requirements to be highly desirable.

2 AdaTape: Adaptive Computation with Elastic Input Sequence

Neural networks can obtain adaptive ability by using different functions or variable computation budgets for different inputs. Assume deep neural network is a function $f(\boldsymbol{x}; \boldsymbol{\theta})$. The output of this function relies on both input $\boldsymbol{x}$ and parameter $\boldsymbol{\theta}$. For adaptive function types, we usually sparsely activate a subset of parameters $\boldsymbol{\theta}$ conditioned on $\boldsymbol{x}$. This type of adaptive ability can also be named as conditional computation. Research on Mixture-of-Experts (Fedus et al., 2021; Lepikhin et al., 2020; Xue et al., 2022; Lou et al., 2021; Riquelme et al., 2021) in fact introduce adaptivity on the function type via routing and specifying the computation for each input example.

Another line of adaptive computation research is dynamic computation budget. For standard neural networks (*e.g.,* transformer), the computation budget is fixed for different samples. However, recent studies show that adaptive computation budgets can help to solve many tasks where vanilla transformer fail (Dehghani et al., 2018; Banino et al., 2021; Abnar et al., 2020). Most of these works use dynamic depth to achieve adaptivity on the allocation of computation budget. For instance, Schuster et al. (2022) employ a mechanism for using different numbers of layers for each input token dur-

ing decoding. The Adaptive Computation Time (ACT) algorithm was proposed by Graves (2016) for an adaptive computational budget on recurrent neural networks (Hochreiter & Schmidhuber, 1997). Universal Transformer (UT) (Dehghani et al., 2018) extends the ACT algorithm to transformers (Vaswani et al., 2017) and makes the computation budget rely on the number of transformer layers used for processing each input example or tokens. Many recent studies like PonderNet (Banino et al., 2021) follow the same framework, while improving the dynamic halting mechanisms.

AdaTape not only uses different function types per input via conditioning the adaptive tape reading mechanism on the input representation, but also adjusts the computation budget by employing variable length memory tape for different inputs. Figure 1 presents a high-level schema of AdaTape in the context of image recognition. For a given batch of input images, after tokenization (e.g., a linear projection of non-overlapping patches from the image in vision transformer), AdaTape uses a vector representing each input to dynamically select a variable-sized sequence of tape tokens from a tape bank. A tape bank could be a set of trainable vectors or generated on-the-fly from inputs, e.g., by using a more fine-grained tokenizer. The selected tape tokens are appended to the original input and fed to the Transformer layers. More precisely, AdaTape defines $f(\boldsymbol{x};\boldsymbol{\theta})$, where $\boldsymbol{x} = \boldsymbol{x}^I : \boldsymbol{x}^T$, and $\boldsymbol{x}^I$ being input tokens and $\boldsymbol{x}^T$ being a sequence of tape tokens that is generated by $f_{\text{ATR}}(\boldsymbol{x}^I, \boldsymbol{Z}_{bank})$. When using a learnable bank, f_{ATR} in fact selects different parameters of the model (learnable tapes) per input meaning adaptivity in terms of the function type. Also given that the length of $\boldsymbol{x}_t$ is different for each input, AdaTape allocates different computation budgets for different inputs. Finally, a sequence containing both input and tape tokens goes to a transformer encoder. For each transformer layer, we use the same multi-head attention across all input and tape tokens. However, we use two different feed-forward networks, one is applied to all tokens from the original input and the other is applied to all tape tokens. We observed slightly better quality by using separate feed-forward networks for input and tape tokens. Note that such a modification can be seen as having two experts, with a fixed routing strategy that has no implication on the speed.

In the following section, we briefly explain the original ACT algorithm that inspires the adaptive token reader mechanism in AdaTape and discuss its shortcomings. Then, we explain different components of AdaTape in detail and share the reasoning behind all the design choices.

2.1 Adaptive Computation Time

In this section, we explain ACT mechanism introduced in (Graves, 2016) and used in UT (Dehghani et al., 2018). As shown in Algorithm 1, ACT uses a trainable linear component $g(\cdot)$ that computes the halting score at each step. When the summation of the halting scores h_p is greater than the halting threshold [1], the computation stops and we conduct early exiting to drop the future layers. At each time step, the states $\boldsymbol{S}$ are weighted by halting score p and accumulated. When the model decides to end the algorithm, the summation of the halting score $p_{sum} = \sum_{t=1}^{T} p_t = 1.0$.

ACT employs a loss function l_{act} to incentives using less computation when it is possible. The loss includes two components, the number of updates, n, and the remainder r. The main goal of ACT is to control the computation by minimizing the number of updates n. However, such an objective is not differentiable and cannot be minimized directly via back-propagation. To partially address that, ACT optimizes n together with the remainder r. When decreasing r, the summation of all used $p_{sum} = \sum_{t=1}^{T} p_t$ increases. When $\sum_{t=1}^{T-1} p_t > 1.0$ - ϵ, n would decrease by 1.0 thus ACT minimizes n indirectly. The final loss function can usually be written as $l = l_{main} + \lambda l_{act}$, where l_{main} represents the downstream task, e.g., cross-entropy loss in classification task. Note that the stability and performance of the model when using ACT is sensitive to the ACT loss coefficient, so, using ACT requires a careful tuning of λ.

2.2 Tape Bank

AdaTape uses a bank of tokens where all the candidate tape tokens are stored. The adaptive tape reader mechanism is independent of where the tokens in the bank come from and simply requires interacting with a $\mathbb{R}^{B\times H}$ tensor, where B is the size of the bank and H is the dimension of each

[1] We usually set the halting threshold as one number close to 1.0, and we use 1.0 - ϵ in algorithm 1. Making the halting threshold slightly smaller than 1.0 is to enable model to halt at the first step.

Algorithm 1 Adaptive Computation Time (ACT)

Input: Initial states $\boldsymbol{S} = \{\boldsymbol{s}_1, ..., \boldsymbol{s}_N\}$, where $\boldsymbol{s}_n$= $\mathbf{0}$; Output states $\boldsymbol{S}^{out} = \{\boldsymbol{s}^o 1, ..., \boldsymbol{s}^o_N\}$, where $\boldsymbol{s}^o_n$= $\mathbf{0}$; Initial halting score $h_p = 0$; Initial update times $n = 0$; Max ponder times T; Trainable transformer layer $g(\cdot)$; Set ϵ as a small number like 0.01; Initial ponder loss l_{atr} = 0.
while $h_p < 1.0$ and $n < T$ **do**
  Compute halting score for each token at this step $\boldsymbol{p}$ = sigmoid$(g(\boldsymbol{S}))$
  Compute the average halting score for each sample p = mean$(\boldsymbol{p})$
  if h_p + $p > 1.0$ - ϵ **then**
    Update remainder r = 1 - h_p
    Update output states $\boldsymbol{S}^{out}$ += $\boldsymbol{S}$ * r
    break
  Increase update times n += 1
  Update states $\boldsymbol{S}$ = $g(\boldsymbol{S})$
  Update output states $\boldsymbol{S}^{out}$ += $\boldsymbol{S}$ * p
  Increase the accumulated halting score h_p += p
l_{act} = n + r
return l_{act}, $\boldsymbol{S}^o$

token. We explore two different ways of creating the tape bank in this paper, an input-driven bank and a learnable bank.

Input-driven Bank For an input-driven bank, we use input data as the source to generate bank tokens on-the-fly. The general idea is to extract a bank of tokens from the input, while employing a different approach than what the model tokenizer uses for mapping the raw input to a sequence of input tokens. This enables dynamic, on-demand access to information from the input that is obtained using a different point of view, e.g., a different resolution or a different level of abstraction. For instance, for the image recognition task, when using Vision Transformers (Dosovitskiy et al., 2020), a simple, yet effective idea is to use different patch sizes for generating input tokens and bank tokens. For instance, using a smaller patch size for generating the bank can be seen as dynamic multi-scale processing of the input where we can select what fine-grained information from the input is useful which is much more efficient than using all the small patches. , which would consume a large amount of computation. Appendix A.2 provides more details on input-driven bank setup for Vision Transformer. In this paper, we use the input-driven bank in one of our image recognition experiments, however, the idea can be generalized to other tasks and setups, e.g., language tasks where more fine-grained tokenization (Kudo & Richardson, 2018) has shown to be effective.

Learnable bank Using a finer grain tokenizer for generating a tape bank is not always applicable. For instance, it is hard to further split each node in graph transformer (Yun et al., 2019). A different and more general approach for generating the tape bank, is to simply use a set of trainable vectors as tokens in a tape bank. The learnable bank can be seen as an embedding layer with a trainable tensor of size $\boldsymbol{Z}_{bank} \in \mathbb{R}^{B \times H}$. One difference between an input-driven bank is, instead of building the bank on-the-fly, the content is fixed for different samples. On the other hand, selecting tape tokens from a learnable bank means partially using parameters of the model, which is a form of sparsity in AdaTape.

2.3 Adaptive Tape Reading Mechanism (ATR)

In order to have elasticity on the input length, we need a mechanism that selects a variable sequence of tape tokens from the bank conditioning on the input representation. Such a mechanism needs to be able on making a decision on how many tape tokens should be added to the input, e.g., using a halting mechanism where tape tokens are recursively added to the input. As the input representation, we can use `CLS` token or the average pooling of all input tokens. This vector is used for both querying the bank or making the decision on the number of tape tokens for each input.

The of-the-shelve ACT algorithm explained in Section 2.1 is based on assumptions that are not desirable in the adaptive token reading scenario. For instance, as shown in algorithm 1, one key requirement in ACT for computing the halting probabilities is to assign different scores to different

Algorithm 2 Adaptive Tape Reading

Input: Initial halting score $h_p = 0$; A list of tokens selected $states = []$; Initial query $\boldsymbol{q} \in \mathbb{R}^{1\times h}$; Tape token bank $\boldsymbol{Z}_{bank} \in \mathbb{R}^{C\times H}$; Token bank mask $\boldsymbol{m}$ = **0**, where $\boldsymbol{m} \in \mathbb{R}^{1\times C}$, where $\boldsymbol{m}$ = **0**; Max ponder times T; Halting threshold τ; Initial ponder loss l_{atr} = 0.
while $h_p < \tau$ **do**
  Compute the inner product of query and tape tokens $\boldsymbol{d} = \boldsymbol{q} \cdot \boldsymbol{Z}_{bank}[:, : h]^T$, where $\boldsymbol{d} \in \mathbb{R}^{1\times C}$
  Mask inner product $\boldsymbol{d} = \boldsymbol{d} \odot (1.0 - \boldsymbol{m})$
  Compute index of top K elements in $\boldsymbol{d}$, set the index vector as $\boldsymbol{i} \in \mathbb{R}^{1\times K}$, where $K = \frac{T}{\tau}$
  Take top K elements from $\boldsymbol{d}$ by the index vector $\boldsymbol{i}$ and denote this new vector as $\boldsymbol{w}$, where $\boldsymbol{w} \in \mathbb{R}^{1\times K}$
  Normalize the elements by softmax: $\boldsymbol{w} = \text{softmax}(\frac{\boldsymbol{w}}{\sqrt{h}})$
  Take top K tape tokens from $\boldsymbol{Z}_{bank}$ by the index vector $\boldsymbol{i}$, and merge the tape tokens $\{\boldsymbol{s}_1, ..., \boldsymbol{s}_K\}$ selected from bank as one single tape token $\boldsymbol{s} = \sum_{k=1}^{K} w_k \boldsymbol{s}_k$
  Append this token into input sequence $states$ += $\boldsymbol{s}$
  if $h_p + \max(\boldsymbol{w}) > \tau$ **then**
    break
  Increase halting score: h_p += $\max(\boldsymbol{w})$
  Increase ponder loss: l_{atr} += 1.0 - $\sum_{k=1}^{K} w_k^2$ or $l_{collect}$ += h_p
  Update token mask by the one-hot vector of $\boldsymbol{i}$: $\boldsymbol{m}$ += sum((one-hot($\boldsymbol{i}$), axis=0)
  Update query $\boldsymbol{q} = \frac{1}{2}(\boldsymbol{s} + \boldsymbol{q})$ or $\boldsymbol{q} = \boldsymbol{s}$
return l_{atr}, $states$

states, where these halting scores should sum up to 1.0.[2] However, such a requirement contradicts the desired behavior in adaptive token reading, as we expect the scores of the already selected tape tokens to stay close to 1.0. Another key contraction is, the existing of layer normalization will make trainable halting score computation layer $g\cdot$ in ACT algorithm invalid. We introduce this reasoning in Appendix A.3. We therefore need to design a new adaptive computation algorithm for elastic input sequences.

Algorithm 2 presents a pseudo code for the Adaptive Tape Reader. ATR is an iterative process for selecting tape tokens from the bank, where at each iteration, we select a new set of tokens (here we select and add tape tokens one-by-one) without replacement, conditioned on the previously selected tokens. ATR uses a query vector $\boldsymbol{q} \in \mathbb{R}^H$ representing the input at the current iteration (i.e., the sequence of all input tokens plus already selected tape tokens) to select the next set of tokens from a tape bank $\boldsymbol{Z}_{bank} \in \mathbb{R}^{B\times H}$. We compute $\boldsymbol{d}$ as the inner product of $\boldsymbol{q}$ and $\boldsymbol{Z}_{bank}$. Note that in practice, to reduce the computation, we can use part of the $\boldsymbol{q}$ and tape tokens for computing the inner product (e.g., $\boldsymbol{q}[: h]$, and $\boldsymbol{Z}_{bank}[:, : h]$, meaning to use the first h elements). To avoid the repeated selection of tape tokens, at each iteration, we adjust the inner product $\boldsymbol{d}$ by masking out weights of tokens that are selected before (using the bank mask $\boldsymbol{m}$ in Algorithm 2 that gets updated in each iteration). Then we select top-K tape tokens according to their scores from $\boldsymbol{d}$ and create $\boldsymbol{w} \in \mathbb{R}^{1\times K}$ containing weight logits of the selected tape tokens in form. We normalize $\boldsymbol{w}$ and apply softmax. We also create the corresponding tape tokens vectors $\{\boldsymbol{s}_1, ..., \boldsymbol{s}_K\}$ collected from the $\boldsymbol{Z}_{bank}$, and compute a single vector $\boldsymbol{s}$ as the weighted average of the selected K tokens, which is the tape token that we add to the input at the current iteration.

To make the halting decision, we accumulate the largest value in $\boldsymbol{w}$ into h_p until it is greater or equal to a threshold τ. Larger $\max(\boldsymbol{w})$ indicate the need for less iteration (i.e., adding fewer tape tokens). Note that the query vector $\boldsymbol{q}$ used for selecting the tape token is updated by averaging the input query and current tape token or just replacing the input query with the current tape token.

In order to incentives shorter sequences for efficiency and penalize the model for adding tape tokens when there is no need, we use a similar loss term to what the original ACT uses, i.e., $l = l_{main} + \lambda l_{atr}$. We observe, Unlike ACT, the adaptive token reader is less sensitive to the value of λ. We also empirically found that using a learnable bank, we can omit the l_{atr} loss term without observing a significant change in the performance of the average length of selected memory tape.

ATR offers a template for elastic input sequences. To emphasize some of the details in the above algorithms, we observe better performance and stability when using a lower dimension for the query and found normalizing the inner product by query dimension $\sqrt{h}$ before applying softmax critical

[2]There is a similar requirement in the halting mechanism from PonderNet (Banino et al., 2021).

following the original design of attention (Vaswani et al., 2017). We also found that normalizing the bank and query using a shared LayerNorm$(\cdot)$ layer improves performance and stabilizes training.

3 EXPERIMENTS

We first present our experimental setup and implementation details in Section 3.1. We report the results on parity task and image classification benchmarks in Section 3.2 and 3.3. To further validate the effectiveness of our model, we ablate the adaptivity, computation cost, and number of trainable parameters in Section 3.4, 3.5 and 3.6. Finally, Section 3.7 provides some insights into the behavior of adaptive tape readers in AdaTape.

3.1 EXPERIMENTAL SETTING

We evaluate AdaTape on both parity task and image recognition benchmarks. First, the parity task is, given a sequence of numbers $\boldsymbol{x} = [x_1, x_2, ..., x_N]$, where x_n is 1, -1, or 0, we are to predict whether the number of 1's in $\boldsymbol{x}$ is even or odd. For the parity task, we employ an input-driven bank and use the bank as the way the model accesses the input information and as the input token, $\boldsymbol{s}$, we simply use a single token vector with random values to create the initial query for tape token reading. We train all models for 10000 steps with batch size 128. The learning rate is set as 3e-5, and we use a linear warm-up for 1000 steps. We use transformer-Tiny in this task since we cannot see improvements when we scale it up, similar to the observation of UT (Dehghani et al., 2018) for algorithmic tasks. The configuration of transformer-Tiny can be found in Appendix A.4. For image classification benchmarks, we conduct large scale pre-training on JFT-300M (Sun et al., 2017) followed by few-shot learning on a wide range of datasets, including ImageNet (Deng et al., 2009), Cifar100 (Krizhevsky et al., 2009) and Pets (Parkhi et al., 2012) following the protocol of Dosovitskiy et al. (2020). We use the same hyper-parameters and model configurations with baselines for a fair comparison, but we believe the performance can be further improved by hyper-parameter tuning. Please see Appendix A.5 for detailed hyper-parameters.

We use the standard transformer (ViT for image classification) and universal transformer (UT) as our baselines. UT is a transformer-based model with adaptive depth. On image classification tasks, as vanilla UT is sharing parameters across layers, it is unfair to compare it directly. We thus develop two improved versions of UT as our strong baselines equipped with adaptive computation. We first consider UT without parameter sharing and refer to it as Unshared Universal Transformer (U2T). We increase the maximum number of ponder steps in U2T, which means U2T has more layers, more computation and more parameters. To further enhance the baselines, we also consider stacking UT and ViT together. We found putting a shallow UT on top of ViT totally failed due to an unstable training process, but putting ViT on top of UT can achieve decent performance. We refer to this model as UViT which is our strongest baseline on image classification with adaptive depth. In UViT, we employ 3 UT layers on Base models and 6 UT layers on Large models. The number of standard transformer layers is the same as ViT.

3.2 EVALUATION ON PARITY TASK

We evaluate AdaTape on the parity task in this section. In some sense, parity is the simplest noncounter-free, or periodic regular language (Bhattamishra et al., 2020). Simple recurrent neural networks can solve this task well because the memory in the recurrect neural network can record the states for finite-state automation Abnar et al. (2021). Standard transformer totally failed in modeling such sequences (Hahn, 2020) as they are incapable of directly maintaining a counter. In AdaTape, we can use the tape tokens to record the behavior in each recurrence step. Specifically, we first use the input sequence $\boldsymbol{x}$ to generate the bank. The Adaptive Token Collection algorithm plays the role to model the parity task recurrently. Considering the target is to predict the number of 1 in the input sequence is even or odd, there are actually two states in the finite-state automation. We then fix the K as 2 in algorithm 2 to check whether there is a state switching after each step. When K is fixed as 2, the maximum ponder time T=2τ=$\frac{N}{2}$, where N is the length of input parity sequence. Such a hyper-parameter setting is to match the period in parity and to ensure AdaTape can collect all required tokens to make the prediction. We take `[CLS]` token representation as query $\boldsymbol{q}$ to select tape tokens.

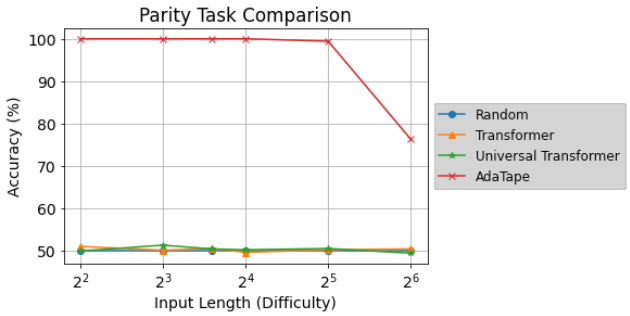

Figure 2: Evaluation on parity task. Standard transformer and universal transformer totally failed on this task, which have a similar accuracy with random guessing.

Table 1: Pre-training and transfer learning results on image classification benchmarks. We report two AdaTape models with different bank types at each scale. AdaTape-B/32-Learn means we are using learnable bank for AdaTape-B/32. AdaTape-B/32-Input denotes we are using input-driven bank. We report both GFLOPs and throughput (measured in images / core / second) for all models. The pre-training is conducted on JFT-300 dataset and we report precision@1 (%) on the validation dataset. The few-shot experiments are on ImageNet, Cifar100, and Pets datasets. We report the Top-1 accuracy on the few-shot evaluation. IN_{25} denotes the result on ImageNet 25-shot.

Model	GFLOPs	Throughput	JFT-300M	IN_{10}	IN_{25}	Cifar100_{10}	Pets_{10}
ViT-B/32	4.437	793.8	43.3	59.4	62.5	72.2	90.3
U2T-B/32	5.513	343.4	38.8	49.5	53.6	64.4	84.1
UViT-B/32	5.511	478.7	44.4	61.9	64.6	74.4	91.6
AdaTape-B/32-Learn	5.585	431.8	44.4	63.0	65.8	75.0	**93.2**
AdaTape-B/32-Input	6.057	342.3	**44.7**	**63.8**	**65.9**	**75.5**	93.0
ViT-B/16	17.634	253.7	48.6	67.4	70.0	77.5	93.0
U2T-B/16	22.016	102.9	46.6	63.0	66.0	73.1	92.5
UViT-B/16	22.011	151.7	49.0	68.5	71.0	75.8	94.0
AdaTape-B/16-Learn	18.837	167.1	**49.1**	**70.3**	**72.6**	**78.7**	94.0
AdaTape-B/16-Input	19.304	148.8	48.9	69.0	71.5	77.4	**94.4**
ViT-L/32	15.628	192.3	49.9	69.5	72.1	79.4	94.0
UViT-L/32	19.242	127.1	**50.5**	70.5	72.6	80.1	94.1
AdaTape-L/32-Learn	19.497	108.6	50.2	70.8	73.3	79.6	**95.0**
AdaTape-L/32-Input	20.141	95.5	50.2	**71.8**	**73.8**	**81.3**	**95.0**
ViT-L/16	61.724	63.1	54.3	75.2	77.0	83.1	95.9
UViT-L/16	77.114	44.7	**54.8**	75.5	77.4	81.6	95.7
AdaTape-L/16-Learn	65.941	44.2	54.7	76.5	78.0	82.7	**96.7**
AdaTape-L/16-Input	66.570	40.1	**54.8**	**76.7**	**78.5**	**84.7**	96.4

Figure 2 shows the results on the parity task. We can see both standard transformer and universal transformer totally failed on parity task, even if we are training with a very short (simple) sequence. However, the AdaTape performs much better than all baselines. One reason is that AdaTape incorporates the recurrence within the model input selection. Although this recurrence is not heavy, it enables AdaTape to model things impossible like maintaining a counter in the standard transformer.

3.3 Evaluation on Image Classification

We also evaluate AdaTape on popular image recognition benchmarks and the results are reported in Table 1. We can see U2T performs worse than other models, we suggest the reason is the unstable training loss when making new halting decisions. The advanced baseline, UViT performs better than ViT because of adaptivity, more computation and parameters. Our AdaTape is using less computation than UViT and performs better at all scales. AdaTape with an input-driven bank performs better at a larger scale than AdaTape with learnable bank. We suggest larger scale model is better at finding informative tokens from input. We can also observe AdaTape is not very good at throughput. Both types of AdaTape are not as hardware friendly as baselines. This is the limitation of AdaTape. However, please note this only means the training speed is slightly slower on TPU, which is

Table 2: Ablation study on the adaptive abilities of AdaTape.

Model	JFT-300M	IN 10-shot	IN 25-shot	Avg SeqLen	Var SeqLen
AdaTape-B/32	44.7	63.8	65.9	57.4	4.8
w/o Ada Length	44.2	63.3	66.4	59.0	0.0
w/o Ada Length & Ada Content	43.9	61.0	64.1	59.0	0.0
AdaTape-B/16	48.9	69.0	71.5	203.4	7.3
w/o Ada Length	49.2	69.4	71.8	206.0	0.0
AdaTape-L/32	50.2	71.8	73.8	56.8	8.5
w/o Ada Length	50.5	71.7	74.2	59.0	0.0
AdaTape-L/16	54.8	76.7	78.5	203.0	10.4
w/o Ada Length	54.5	76.7	78.4	206.0	0.0

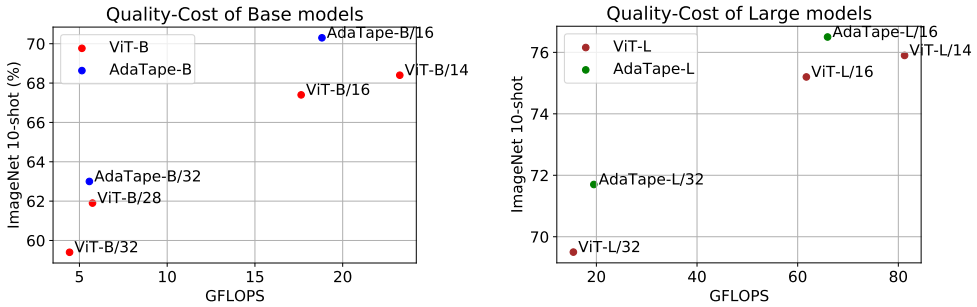

Figure 3: Ablation Study on quality-cost trade-off. We scale the standard transformer (*i.e.,* ViT) to smaller patch size, *e.g.,* ViT-B/14 and ViT-L/14, and compare it with AdaTape.

highly optimized by large-scale matrix multiplication. Due to less real computation (GFLOPS) and dynamic sequence length, AdaTape has the potential to speedup inference on other hardwares.

3.4 Ablation on Adaptive Ability

The adaptive abilities of AdaTape include adaptive sequence content and adaptive sequence length. We invesitgate the contribution of these two adaptive abilities in this section. For adaptive sequence content, we expect it can improve effectiveness. For the adaptive sequence length, as it has the potential to save compute during inference, we expect AdaTape can keep a comparable accuracy when we use the fixed sequence length.

Adaptive sequence length is from ATR algorithm with recurrent token selection process. The adaptive sequence content is mainly from selective use of the tape bank. To remove the adaptive length, we select all T tokens in parallel and then spend the same amount of computation cost for all samples. Different from adaptive sequence length, removing adaptive content only is impossible, because we our ATR algorithm is required to have a tape bank. Therefore, we remove adaptive length and content together by appending a fixed set of trainable tokens to every input sample.

Results are shown in Table 2. We can see, without the adaptive content, there is a significant performance drop. For instance, without adaptive content, compared with AdaTape without adaptive sequence only, AdaTape-B/32 degrades 2.3 points on ImageNet 10-shot accuracy. This shows the effectiveness of adaptive sequence content. Without adaptive computation, this shows the upper bound of AdaTape. Under this setting, when we remove the adaptive sequence length, we can see there is a comparable performance instead of a significant improvement at different scales, which shows the tape tokens selected are condensed and making full use limited input. Another interesting finding is the variance of sequence length. When we scale the model up, we can observe larger models has larger variance. This indicates larger AdaTape is better at controlling sequence length and computation cost.

Table 3: 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

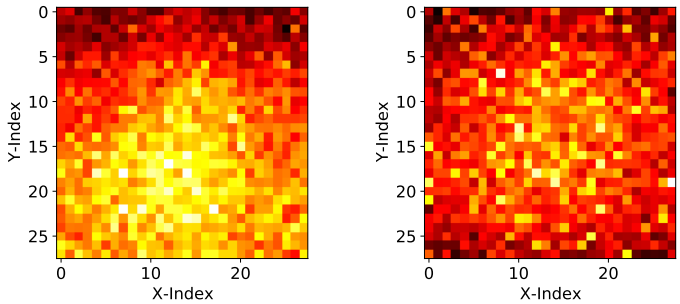

Figure 4: We visualize the tape token selection heatmap of AdaTape-B/32 (left) and AdaTape-B/16 (right). The hotter color means the patch at this position is more frequently selected.

3.5 Ablation on Computation Cost

Although AdaTape only increases the computation cost slightly, to further verify the improvement is from more reasonable design and adaptive abilites instead of more computation, we conduct ablation experiments on computaiton cost. To this end, we scale vanilla transformer with smaller patch size. As shown in Figure 4, although smaller patch size can improve ViT, AdaTape is still outperforming ViT with less computation.

3.6 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 only from more parameters, we increase the 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 3. We can see AdaTape outperforms ViT with 3 FFNs, although the ViT-B/28-3FFN and ViT-B/14-3FFN are using more computation and more parameters. For ViT-B/14-3FFN, AdaTape-B/16 surpasses it by 1.3% in terms of top-1 accuracy on ImageNet 10-shot.

3.7 Visualization

We study the behavior of AdaTape in this section. First, we collect the token selection results of AdaTape with input-driven bank on JFT-300M evaluation sets, and visualize as heatmaps in Figure 4. We can see the central patches are more frequently picked (with lighter color). This matches well with our prior knowledge, central patches are usually more informative. This shows AdaTape prefers more informative patches to enhance the performance. We also visualize the token selection distribution. Please see Appendix A.7 for details.

4 Conclusion

We propose AdaTape, a novel strategy and general purpose adaptive computation method. AdaTape is characterized by elastic sequence lengths generated from a newly proposed Adaptive Tape Reader mechanism. This new inductive bias enables AdaTape to have the potential to solve challenging tasks that standard transformer and existing adaptive architecture transformer struggle at. Via extensive experiments on image recognition benchmarks, we show that AdaTape outperforms standard transformer and adaptive architecture transformer in a compute-matched setup.

REFERENCES

Samira Abnar, Mostafa Dehghani, and Willem Zuidema. Transferring inductive biases through knowledge distillation. *arXiv preprint arXiv:2006.00555*, 2020.

Samira Abnar, Mostafa Dehghani, Behnam Neyshabur, and Hanie Sedghi. Exploring the limits of large scale pre-training. *arXiv preprint arXiv:2110.02095*, 2021.

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the ability and limitations of transformers to recognize formal languages. *arXiv preprint arXiv:2009.11264*, 2020.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Mostafa Dehghani, Anurag Arnab, Lucas Beyer, Ashish Vaswani, and Yi Tay. The efficiency misnomer. *arXiv preprint arXiv:2110.12894*, 2021.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2021.

William Fedus, Jeff Dean, and Barret Zoph. A review of sparse expert models in deep learning. *arXiv preprint arXiv:2209.01667*, 2022.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171, 2020.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 66–71, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-2012. URL https://aclanthology.org/D18-2012.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Yuxuan Lou, Fuzhao Xue, Zangwei Zheng, and Yang You. Sparse-mlp: A fully-mlp architecture with conditional computation. *arXiv e-prints*, pp. arXiv–2109, 2021.

David Meunier, Renaud Lambiotte, Alex Fornito, Karen Ersche, and Edward T Bullmore. Hierarchical modularity in human brain functional networks. *Frontiers in neuroinformatics*, 3:37, 2009.

Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems*, 34:8583–8595, 2021.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *arXiv preprint arXiv:2207.07061*, 2022.

Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pp. 843–852, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Fuzhao Xue, Ziji Shi, Futao Wei, Yuxuan Lou, Yong Liu, and Yang You. Go wider instead of deeper. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 8779–8787, 2022.

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12104–12113, 2022.

A APPENDIX

A.1 INPUT-DRIVEN BANK DETAILS

We introduce more details about 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.

A.2 SCALING WITH PATCH SIZE

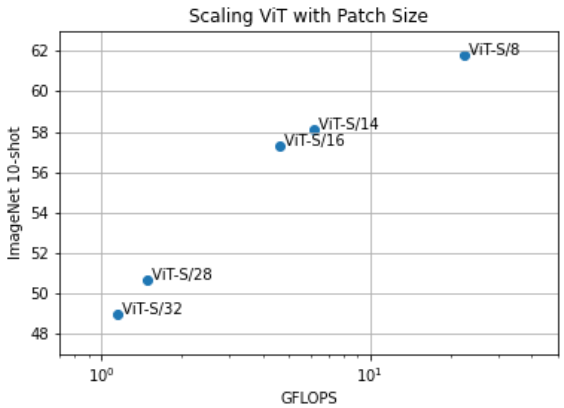

Figure 5: We scale ViT with patch size. The model is pre-trained on JFT-300M (Sun et al., 2017). We report the few-shot performance on ImageNet (Deng et al., 2009)

We fix all other parameters and scale ViT with patch size in Figure 5. We select ViT-S as platform as longer sequence is expensive. We cannot train ViT-B/8 with data parallelism on 512 TPUv3 cores due to the memory constraint. In Figure 5, when adding more computation, there is a significant accuracy improvement. Such improvement inspires us to use smaller patches as the source to generate bank.

A.3 LAYERNORM IS MAKING ACT INVAILD IN ADAPTIVE SEQUENCE

We argue that layer normalzation in transformer is making ACT algorithm invalid. We first assume we using a transformer with following 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 tth 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 not rigorous but 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 find p_t is removed during the normalization, so the value of p_t cannot change the output of the first $\mathrm{LayerNorm}(\cdot)$ in practice, and then will make effect on the following layers. That means, 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. One possible solution is assigning the weights to tape tokens after normalization layers. This is a simple solution may make sense. However, we observe the weights of all tokens will increase to 1.0 very fast and the model will always only append one token even if we do not use any loss function to regularize p_t. This is not expected for two reasons. First, there is no adaptive ability observed. Second, the training is also extremely unstable under this design.

In summary, we cannot assign weights to different tokens. That means we cannot have a trainable linear layers to compute the halting score p. That makes the ACT-based dynamic halting mechanism, including PonderNet, not applicable. We therefore are required to design a new adaptive computation algorithm for elastic input sequence.

A.4 Configuration for Transformer-Tiny

Table 4: 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 parity task. As shown in Table 4, Tiny level transformer still use 12 layers, which is the same as transformer base. The difference is that tiny configuration has smaller hidden dimension and fewer attention heads.

A.5 Hyper-parameters

Table 5: Hyper-parameters for AdaTape on image classification

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

On vision tasks, we follow Dosovitskiy et al. (2020) and train all models for 7 epochs on JFT-300M. We use the same learning rate, batch size and learning schedule. Customized hyper-parameters for AdaTape is summerized in Table 5. We use the fixed max ponder times for all models. Smaller τ on AdaTape-L with 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. Also, AdaTape with learnable bank can be trained without halting loss.

A.6 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 token is controlled by the model adaptively so to control the real length directly, we use the AdaTape without adaptive length as platform. We sweep the number of tape tokens T over $\{5, 10, 20, 40\}$ and summerize 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 longer sequence means more computation budget, we select to use 10 tape tokens as the default choice.

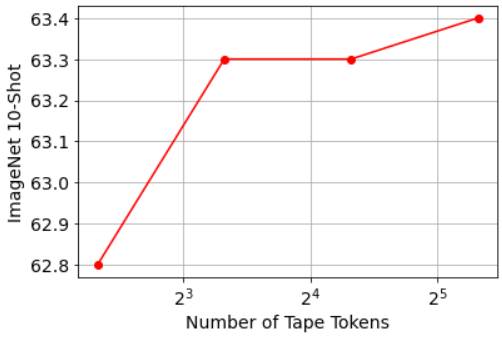

Figure 6: We sweep the number of tape tokens over {5, 10, 20, 40}. Considering more tape tokens means much more computation cost, we set 10 as our default choice.

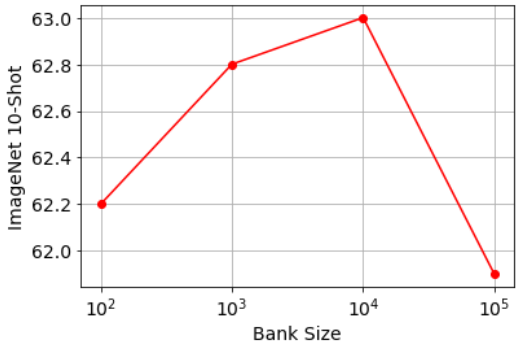

Figure 7: We sweep the bank size over {1e+2, 1e+3, 1e+4, 1e+5} for AdaTape with learnable bank, and found 1e+4 is the sweep point.

Bank Size We also sweep the bank size over {5, 10, 20, 40} for AdaTape with learnable bank. As shown in Figure 7, the AdaTape performs best when we have 1e+4 tape tokens in the bank. As we only conduct 10 steps of recurrence by default, larger bank will not increase the computation cost significantly.

A.7 Visualization of Token Selection Distribution

Similar to Section 3.7, we collect the token selection results on JFT-300M evaluation set. We sort the tape token index by the frequency and visualize the token selection distribution in Figure 8. We can observe the token selection decision obey long-tail distributions. That shows our AdaTape prefers the tape tokens at some specific positions, which matches well with our observation in Figure **??**, *i.e.,* central patches are frequently selected.

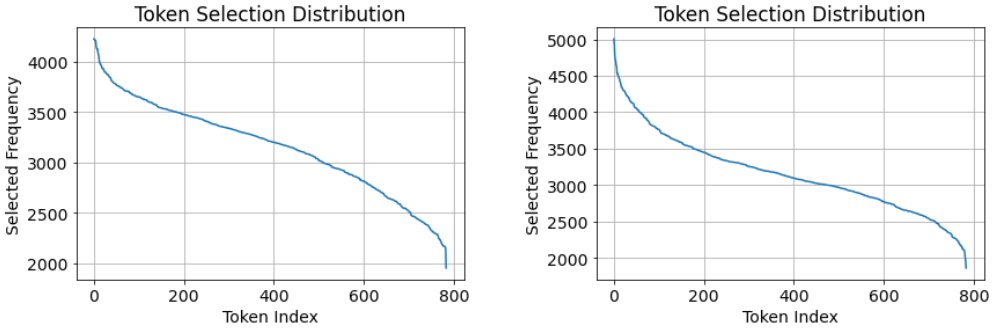

Figure 8: We visualize the tape token selection distribution in AdaTape-B/32 (left) and AdaTape-B/16 (right). The index id is sorted by the value of selected frequency