RIPPLE ATTENTION FOR VISUAL PERCEPTION WITH SUB-QUADRATIC COMPLEXITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformer architectures are now central to modeling in natural language processing tasks. At its heart is the attention mechanism, which enables effective modeling of long-term dependencies in a sequence. Recently, transformers have been successfully applied in the computer vision domain, where 2D images are first segmented into patches and then treated as 1D sequences. Such linearization, however, impairs the notion of spatial locality in images, which bears important visual clues. To bridge the gap, we propose *ripple attention*, a sub-quadratic attention mechanism for visual perception. In ripple attention, contributions of different tokens to a query are weighted with respect to their relative spatial distances in the 2D space. To favor correlations with vicinal tokens yet permit long-term dependencies, we derive the spatial weights through a stick-breaking transformation. We further design a dynamic programming algorithm that computes weighted contributions for all queries in linear observed time, taking advantage of the summed-area table and recent advances in linearized attention. Extensive experiments and analyses demonstrate the effectiveness of ripple attention on various visual tasks.

1 INTRODUCTION

The transformer architecture (Vaswani et al., 2017) has been dominant in various important natural language processing (NLP) tasks, including machine translation (Vaswani et al., 2017; Dehghani et al., 2019), language understanding (Devlin et al., 2018), language modeling (Dai et al., 2019; Baevski & Auli, 2019) and many others. The cornerstone of a transformer is the attention mechanism (Bahdanau et al., 2014) which computes pair-wise interactions between any token pairs of the input sequence. As a result, it is capable of modeling long-term dependencies in a sequence, which is an important factor to the success of transformers.

Recently, the transformer architecture has also found its applications in the domain of computer vision (CV). It is adopted for image classification (Dosovitskiy et al., 2020; Touvron et al., 2020; Liu et al., 2021a; Yang et al., 2021; Wang et al., 2021b), segmentation (Wang et al., 2020b; Strudel et al., 2021), low-level image processing (Chen et al., 2020), image generation (Parmar et al., 2018), object detection (Carion et al., 2020; Meng et al., 2021) and many other tasks. In these vision applications, a 2D image is represented as a set of patches flattened into a 1D sequence. These patches are analogous to the tokens in sequence modeling tasks that are commonly seen in NLP. Nevertheless, such linearization undermines the inherent local structure of a 2D image, which bears important visual clues (Simoncelli & Olshausen, 2001). There often exist strong correlations within local neighborhoods in an image. Therefore, paying more attention to patches in a closer region could facilitate gathering information that is particularly useful in visual pattern recognition. This is similar to the concept of *context* in NLP, just that the structural context of a visual token is scattered in the 1D sequence, making it difficult for the transformer to capture such prior knowledge. In contrast, the convolutional neural network (CNN) (Fukushima & Miyake, 1982; LeCun et al., 1989; Krizhevsky et al., 2012), which has been the de-facto architecture in computer vision tasks for decades, utilizes local receptive fields and achieves good performance. The drawback of that is, as convolution operations are limited to small receptive fields, they have great difficulty in extracting global image features.

Therefore, it is appealing to incorporate the notion of spatial vicinity into the transformer, while still preserving its capacity of modeling long-term dependencies. To bridge the gap, we propose

ripple attention (Figure 1; §3), a novel attention mechanism for visual perception. In ripple attention, contributions from different tokens to a query are weighted with respect to their relative spatial distances in the 2D space. These spatial weights are derived through a stick-breaking transformation (§3.2), which promotes local correlations by leaning to assign larger weights to spatially closer tokens. We then design a dynamic programming algorithm (§3.3) that is capable of executing ripple attention in linear observed time, taking advantage of the recently proposed linearized attention (§2.2) and the summed-area table technique (§3.3).

We validate our method by conducting extensive experiments on image classification and object detection tasks (§4). Ripple attention significantly improves the accuracy of the original vision transformer in image classification and performs competitively with detection transformers for object detection (§4.2), in asymptotically faster runtime (§5.2). Further analysis on the rippling distance and ablation studies (§5.1) indicate that ripple attention favors contributions from tokens in the vicinity yet preserves global information from long-term dependencies.

2 PRELIMINARY

2.1 ATTENTION MECHANISM

Let $\boldsymbol{Q} \in \mathbb{R}^{N \times D}$ denote a set of N query vectors, which attend to M key and value vectors, denoted by matrices $\boldsymbol{K} \in \mathbb{R}^{M \times D}$ and $\boldsymbol{V} \in \mathbb{R}^{M \times C}$ respectively. For a query vector at position n, the softmax attention function computes the following quantity[1]:

$$\mathrm{Attn}\left(\boldsymbol{q}_n, \boldsymbol{K}, \boldsymbol{V}\right) = \sum_{m=1}^{M} \frac{\exp\left(\boldsymbol{q}_n^\top \boldsymbol{k}_m\right)}{\sum_{m'=1}^{M} \exp\left(\boldsymbol{q}_n^\top \boldsymbol{k}_{m'}\right)} \boldsymbol{v}_m^\top \coloneqq \mathrm{softmax}(\boldsymbol{K}\boldsymbol{q}_n)^\top \boldsymbol{V}, \tag{1}$$

which is an average of the set of value vectors $\boldsymbol{V}$ weighted by normalized similarity between different queries and keys. However, such quantity requires computing the similarity between all pairs of queries and keys, incurring quadratic complexity in both time and memory. It makes the computational overhead for long sequences prohibitive, especially in the case of vision tasks.

2.2 LINEARIZED ATTENTION

To reduce the computational complexity in the attention mechanism, prior works propose to linearize the softmax kernel (Choromanski et al., 2020; Katharopoulos et al., 2020; Peng et al., 2021). In particular, they replace the exponential kernel used in softmax functions $\kappa(\boldsymbol{q}, \boldsymbol{k}) \coloneqq \exp\left(\boldsymbol{q}^\top \boldsymbol{k}\right)$ with a dot product of two feature maps $\phi(\boldsymbol{q})^\top \phi(\boldsymbol{k})$, where $\phi(\cdot) : \mathbb{R}^D \to \mathbb{R}^{D'}$. For example, in Katharopoulos et al. (2020) they define $\phi(\boldsymbol{x}) = \mathrm{elu}(\boldsymbol{x}) + 1$, which works well on various tasks.[2] With the defined feature map, linearized attention can be written as:

$$\begin{aligned}\mathrm{LinearAttn}\left(\boldsymbol{q}_n, \boldsymbol{K}, \boldsymbol{V}\right) &\coloneqq \sum_{m=1}^{M} \frac{\phi(\boldsymbol{q}_n)^\top \phi(\boldsymbol{k}_m)}{\sum_{m'=1}^{M} \phi(\boldsymbol{q}_n)^\top \phi(\boldsymbol{k}_{m'})} \boldsymbol{v}_m^\top \\ &= \frac{\phi(\boldsymbol{q}_n)^\top \sum_{m=1}^{M} \phi(\boldsymbol{k}_m) \boldsymbol{v}_m^\top}{\phi(\boldsymbol{q}_n)^\top \sum_{m'=1}^{M} \phi(\boldsymbol{k}_{m'})}.\end{aligned} \tag{2}$$

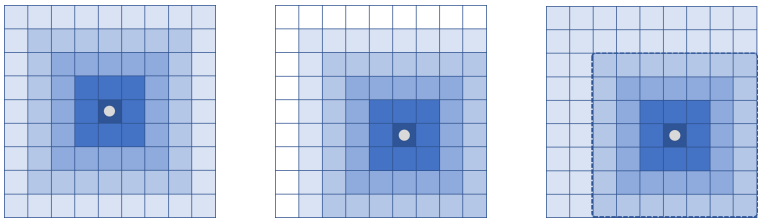
In other words, by grouping together the computations of keys and values, their statistics can be shared for all queries. It therefore achieves linear complexity in both time and memory with respect to the length of the sequence, as we only need to compute $\sum_{m=1}^{M} \phi(\boldsymbol{k}_m)\boldsymbol{v}_m^\top$ and $\sum_{m=1}^{M} \phi(\boldsymbol{k}_m)$ once and then reuse them for each query.

3 MODEL

In this section, we introduce ripple attention, a novel attention mechanism that features the relative spatial vicinity. We start from a reformulation of the linearized attention (§2.2) under the notation

[1] We omit the scaling factor for simplicity.

[2] These feature maps could also be random in the sense that they can also depend on a set of random variables ω; however, throughout our experiments we use deterministic ones as they perform better in vision tasks. Further details and discussions about the choice of feature maps can be found in Appendix C.1.



(a) (b) (c)

Figure 1: A demonstration of vicinal groups in ripple attention on a 9×9 image $\boldsymbol{I}$. Each square denotes a token, the circle denotes the query position and we use deeper color to indicate a larger spatial weight. **Left (a)**: an example of vicinal group partitioning in the case where the query lies in the center of an image, resulting in a symmetric rippling effect over the 2D space; **Middle (b)**: another example of a group partitioning on the same image but the query token is not centered. In this case, distal vicinal groups (the top-left corner) receives almost no spatial weights. **Right (c)**: the same case as in (b) with the threshold τ set such that $\hat{r} = 4$. Groups beyond $\hat{r} - 1$ (indicated by the dashed line) contribute according to an equal spatial weight.

of vicinal groups (§3.1). This reformulation makes it straightforward to introduce a spatial weight associated with each vicinal group, which is the cornerstone of ripple attention. We then describe the derivation of these spatial weights through a stick-breaking transformation (§3.2) and a dynamic programming algorithm (§3.3) based on the summed-area table technique to perform the computation of ripple attention efficiently.

3.1 Ripple Attention

We assume an input image consists of $H \times W$ patch tokens. Given a query token at position (i, j), we partition the whole set of patch tokens into $R + 1$ vicinal groups $\{\mathcal{N}_r(i, j)\}_{r=0}^{R}$, according to their Chebyshev distances r from the position to the query[3], which means for every token at position $(m, n) \in \mathcal{N}_r(i, j)$ we have $\max(|m - i|, |n - j|) = r$. Illustrations of such vicinal group partitioning can be found in Figure 1a or Figure 1b, where each group is marked by a different color.

Under the notation of vicinal groups, we can reformulate the linearized attention as:

$$\text{LinearAttn}\left(\boldsymbol{q}_{ij}, \boldsymbol{K}, \boldsymbol{V}\right) = \frac{\phi(\boldsymbol{q}_{ij})^\top \sum_{r=0}^{R} \sum_{(m,n)\in\mathcal{N}_r(i,j)} \phi(\boldsymbol{k}_{mn}) \boldsymbol{v}_{mn}^\top}{\phi(\boldsymbol{q}_{ij})^\top \sum_{r=0}^{R} \sum_{(m',n')\in\mathcal{N}_r(i,j)} \phi(\boldsymbol{k}_{m'n'})}. \tag{3}$$

This formulation is computationally equivalent since the summations over $\phi(\boldsymbol{k})\boldsymbol{v}^\top$ and $\phi(\boldsymbol{k})$ also cover all positions within the image. The essence of ripple attention is to let tokens respond differently to a query, according to their relative distances. Typically, tokens close to the query in the 2D space should weigh more than the tokens far away in general, since there exist strong local correlations in images. This control is translated into a spatial weight $\alpha_r(i, j)$ associated with each vicinal group $\mathcal{N}_r(i, j)$ and can be easily introduced into linearized attention[4]:

$$\text{RippleAttn}\left(\boldsymbol{q}_{ij}, \boldsymbol{K}, \boldsymbol{V}\right) \coloneqq \frac{\phi(\boldsymbol{q}_{ij})^\top \sum_{r=0}^{R} \alpha_r(i,j) \sum_{(m,n)\in\mathcal{N}_r(i,j)} \phi(\boldsymbol{k}_{mn}) \boldsymbol{v}_{mn}^\top}{\phi(\boldsymbol{q}_{ij})^\top \sum_{r=0}^{R} \alpha_r(i,j) \sum_{(m',n')\in\mathcal{N}_r(i,j)} \phi(\boldsymbol{k}_{m'n'})}. \tag{4}$$

We define $\alpha_r(i, j) \in (0, 1), \forall r = 0, 1, \ldots, R$ and $\sum_{r=0}^{R} \alpha_r(i, j) = 1$, to reweigh contributions of different vicinal groups with respect to a query in the attention computation. By respecting the spatial structure of images, ripple attention constructs a structured context for queries, which facilitates the model to reconcile both global and local information. The name *ripple attention* comes from its similarity to the ripples on the surface of water (Figure 1).

[3]Chebyshev distances is also known as chessboard distance, or more generally L_∞ metric, since it equals $\lim_{p\to\infty} \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$ for any n-dimensional vectors $\boldsymbol{x}, \boldsymbol{y}$.

[4]The partitioning is hard to implement in softmax attention mechanism. More discussions about ripple-softmax and its complexity can be found in Appendix A.

3.2 SPATIAL WEIGHTS

To derive the spatial weights $\alpha_r \in (0,1)\ \forall r = 0, 1, \ldots, R$,[5] we first define a sequence of scalars $\{\mathrm{s}_r\}$, where $\mathrm{s}_r \in (0,1)\,\forall r = 1, \ldots, R$. The spatial weights are parameterized as follows (with $\mathrm{s}_{R+1} = 1$):

$$\alpha_r = \begin{cases} \mathrm{s}_1, & \text{if } r = 0 \\ \mathrm{s}_{r+1} \prod_{r' \leq r} (1 - \mathrm{s}_{r'}), & \text{otherwise} \end{cases} \tag{5}$$

The sequence $\{\mathrm{s}_r\}$ is generated through a small neural network followed by a sigmoid function. See Appendix C.3 for more details. Our construction is analogous to the stick-breaking process in Bayesian nonparametrics (Wasserman, 2006), except that s_r here is a deterministic scalar rather than a random variable. One of its appealing properties is:

$$\sup \left\{ \mathrm{s}_{r+1} \prod_{i \leq r} (1 - \mathrm{s}_i) : \mathrm{s}_{r+1} \in (0,1) \right\} \geq \sup \left\{ \mathrm{s}_{r'+1} \prod_{i \leq r'} (1 - \mathrm{s}_i) : \mathrm{s}_{r'+1} \in (0,1) \right\} \text{ if } r < r'.$$

In other words, we only assume the *supremum* of each spatial weight α_r is decreasing. A stronger constraint could be monotonicity, for example $\alpha_r > \alpha_{r'}$ if $r < r'$. We argue that the former is more favorable, because it offers the flexibility to let distant tokens outweigh when necessary; the effective modeling of such long-term dependencies is deemed as the key to the success of the transformer architecture as well.

In theory, the stick-breaking transformation produces $R + 1$ different spatial weights. However, as the weights become trivially small near the end of this transformation, the computational overhead incurred by them becomes worthless. Therefore, we define a threshold τ to adaptively terminate the transformation at vicinal group $\mathcal{N}_{\hat{r}}(i,j)$ when the length of remaining stick is less than τ (i.e., $1 - \sum_{r=0}^{\hat{r}} \alpha_r(i,j) < \tau$). We then merge all vicinal groups $\mathcal{N}_r(i,j)$ with $r \geq \hat{r}$ and share the same weights among them assuming they contribute equally:

$$\alpha_r(i,j) = \frac{1 - \sum_{r'=0}^{\hat{r}-1} \alpha_{r'}(i,j)}{R - \hat{r} + 2}, \text{ if } r >= \hat{r}. \tag{6}$$

This truncating-and-merging operation is demonstrated in Figure 1c, where the threshold τ is set such that $\hat{r} = 4$. In this case, all the remaining groups (outside the dashed line) share the same weight according to equation 6. Compared to Figure 1b, adaptive ripple allows a stop of the transformation before hitting the boundary, which prevents potentially worthless computations for distal groups. At the same time, it does weigh in the contributions from those groups, preserving the ability to capture long-term dependencies.

3.3 DYNAMIC PROGRAMMING

The only problem left now is how to compute ripple attention effectively. A naïve implementation of equation 4 has a time complexity of $\mathcal{O}(HWR^2)$. Since R is bounded by $\max(H, W) - 1$, the computation is quadratic with respect to the length of the sequence. We give detailed derivations of runtime complexity of each attention variant in Appendix A.

In this section, we present a dynamic programming algorithm built on the summed-area table (SAT) technique, a classic algorithm in computer graphics and computer vision (Crow, 1984; Viola et al., 2001), which reduces the time complexity of ripple attention to $\mathcal{O}(HWR)$. SAT is an efficient data structure that stores prefix sums for each pixel position of an image such that summations over any window in the image can be retrieved in constant time. For an image $\boldsymbol{I}$ with height H and width W, it first initializes the table by computing the cumulative sum $\boldsymbol{S}$ of all the tokens above and to the left of (i,j) inclusively in the 2D plane[6]:

$$\boldsymbol{S}(i,j) = \sum_{i'=1}^{i} \sum_{j'=1}^{j} \boldsymbol{I}(i',j'). \tag{7}$$

[5] In this section, we sometimes drop the dependence on position (i,j) for spatial weights $\alpha_r(i,j)$ when there is no ambiguity.

[6] In practice, we adopt a linear-complexity implementation which first performs the cumulative summation over the row axis and then the column axis, yielding the same result.

For a square window with center (i, j) and radius r (i.e., a region centered at (i, j) with both its height and width equal to $2r + 1$), the summation over its elements is denoted by $\mathcal{W}(i, j, r)$ and can be computed in constant time (Crow, 1984):

$$\mathcal{W}(i,j,r) \coloneqq \boldsymbol{S}(i+r,j+r) - \boldsymbol{S}(i-r-1,j+r) - \boldsymbol{S}(i+r,j-r-1) + \boldsymbol{S}(i-r-1,j-r-1). \quad (8)$$

In this work, we consider $\{\phi(\boldsymbol{k}_{ij})\boldsymbol{v}_{ij}^\top\}$ and $\{\phi(\boldsymbol{k}_{ij})\}$ as generalized pixels within the input image and construct two SATs $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ to compute their prefix summations respectively. According to equation 8, the window sums can be obtained efficiently and are denoted as $\mathcal{W}_1$ and $\mathcal{W}_2$.

We show that the sum of $\phi(\boldsymbol{k})\boldsymbol{v}^\top$ and $\phi(\boldsymbol{k})$ over vicinal group $\mathcal{N}_r(i, j)$ can also be computed within constant time from SATs:

$$\sum_{(m,n)\in\mathcal{N}_r(i,j)} \phi(\boldsymbol{k}_{mn})\boldsymbol{v}_{mn}^\top = \mathcal{W}_1(i,j,r) - \mathcal{W}_1(i,j,r-1);$$
$$\sum_{(m,n)\in\mathcal{N}_r(i,j)} \phi(\boldsymbol{k}_{mn}) = \mathcal{W}_2(i,j,r) - \mathcal{W}_2(i,j,r-1).$$

Intuitively, this can be viewed as taking the difference between the largest square window wrapped by the group and the smallest square window containing the group. Equipped with SATs, the formulation of ripple attention becomes:

$$\text{RippleAttn}\left(\boldsymbol{q}_{ij}, \boldsymbol{K}, \boldsymbol{V}\right) = \frac{\phi(\boldsymbol{q}_{ij})^\top \sum_{r=0}^{R} \alpha_r(i,j)\left(\mathcal{W}_1(i,j,r) - \mathcal{W}_1(i,j,r-1)\right)}{\phi(\boldsymbol{q}_{ij})^\top \sum_{r=0}^{R} \alpha_r(i,j)\left(\mathcal{W}_2(i,j,r) - \mathcal{W}_2(i,j,r-1)\right)}. \quad (9)$$

In §3.2, we merge all groups $\mathcal{N}_r(i, j)$ with $r >= \hat{r}$ assuming equal contributions (equation 6), which can be jointly computed in constant time using $\boldsymbol{S}(H, W) - \mathcal{W}(i, j, \hat{r} - 1)$. Therefore, given a reasonable hyper-parameter choice of τ, the algorithm can achieve linear observed time in the sequence length. This is due to the fact that after the precomputation of SATs (in linear complexity), for each query the required summations of vicinal groups can be computed in constant time. In practice, we do not even need to materialize the tensor $\mathcal{W}$, due to the fact that previously computed results for close vicinal groups could be reused for distant vicinal groups. Therefore, the space complexity of ripple attention remains to be $\mathcal{O}(HW)$ irrespective of the rippling distance R. Algorithm 1 sketches the dynamic programming for the ripple attention mechanism given a single query and a threshold τ.

Efficient gradient computation. The algorithm discussed above addresses the runtime complexity of the forward pass of ripple attention. In Appendix B, we also present a dynamic programming algorithm to compute gradients for the backward pass, again in $\mathcal{O}(HWR)$ time and space complexity. The main idea is to utilize the symmetry of vicinal groups and reformulate the gradient calculations as summations over different groups, where computations could be further reduced using SATs; in contrast, a naïve implementation would come with $\mathcal{O}(HWR^2)$ complexity.

4 Experiments

We conduct extensive experiments on image classification and detection tasks to demonstrate the effectiveness of ripple attention.

4.1 Experimental Setup

Datasets For image classification, we evaluate our model on standard benchmark datasets: (1) `ImageNet1k` dataset (Deng et al., 2009), consisting of approximately 1,280K/50K images of 1000 classes for training/validation splits respectively; (2) `CIFAR-100` (Krizhevsky et al., 2009), which contains 50K images of 100 classes for training and 10K for evaluation. For detection tasks, we conduct our experiment on the `COCO` benchmark (Lin et al., 2014) consisting of 118k training and 5k validation images respectively.

Baselines Our model for image classification is based on the vision transformer architecture (ViT) (Dosovitskiy et al., 2020; Touvron et al., 2020), where the attention block is replaced with ripple attention. We compare ripple attention (referred to as RIPPLE hereafter) with various attention mechanisms in ViT:

Algorithm 1 Dynamic Programming for Ripple Attention

Input: the key-value statistics $I \in \mathbb{R}^{H\times W}$, query position (i, j), spatial weights $\{\alpha_r(i, j)\}$ and threshold τ;
Output: The quantity $\texttt{result} \coloneqq \sum_{r=0}^{R} \alpha_r(i, j)\,(\mathcal{W}(i, j, r) - \mathcal{W}(i, j, r-1))$;
Initialize: $\texttt{result} \leftarrow 0,\ \mathcal{W}_{\text{cur}} \leftarrow 0,\ \mathcal{W}_{\text{prev}} \leftarrow 0,\ S \in \mathbb{R}^{H\times W} \leftarrow \mathbf{0}$;
Compute: summed-area table S by calling `cumsum()` function twice over horizontal and vertical directions respectively;
Compute: $\hat{r}$ by summing over $\alpha_r(i, j)$ until $1 - \sum_{r=0}^{\hat{r}} \alpha_r(i, j) < \tau$;
for $r = 0, 1, \ldots, \hat{r} - 1$ **do**
  $\boldsymbol{S}_1 \leftarrow \boldsymbol{S}(i + r, j + r)$; ▷ Cumulative sum at the bottom-right corner
  $\boldsymbol{S}_2 \leftarrow \boldsymbol{S}(i - r - 1, j + r)$; ▷ Cumulative sum at the bottom-left corner
  $\boldsymbol{S}_3 \leftarrow \boldsymbol{S}(i + r, j - r - 1)$; ▷ Cumulative sum at the top-right corner
  $\boldsymbol{S}_4 \leftarrow \boldsymbol{S}(i - r - 1, j - r - 1)$; ▷ Cumulative sum at the top-left corner
  $\mathcal{W}_{\text{cur}} \leftarrow \boldsymbol{S}_1 - \boldsymbol{S}_2 - \boldsymbol{S}_3 + \boldsymbol{S}_4$;
  $\texttt{result} \leftarrow \texttt{result} + \alpha_r(i, j)(\mathcal{W}_{\text{cur}} - \mathcal{W}_{\text{prev}})$;
  $\mathcal{W}_{\text{prev}} \leftarrow \mathcal{W}_{\text{cur}}$;
end for
$\texttt{result} \leftarrow \texttt{result} + \alpha_{\hat{r}}(i, j)(\boldsymbol{S}(H, W) - \mathcal{W}(i, j, \hat{r} - 1))$;
return result

- DEIT, which adopts the same architecture as ViT and vanilla softmax attention.[7]
- CONVIT, which imposes a soft convolutional inductive bias on the vanilla attention mechanism.
- DEIT-LA, a DEIT model equipped with linearized attention (§2.2) instead of softmax attention. We also include several variants that improve DEIT-LA, such as PERMUTEFORMER (Chen, 2021), SPE (Liutkus et al., 2021) and Rotary positional embeddings (ROPE, Su et al., 2021) that incorporates relative positional encodings.

For object detection, we evaluate our model in the general framework of detection transformer (DETR; Carion et al., 2020) to test the generalization ability of RIPPLE. However, due to the slow convergence of DETR, we are unable to run DETR model for a full training schedule given limited computational resources. Instead, we adopt SMCA (Gao et al., 2021) as our baseline, a variant of DETR that greatly speeds up the convergence by constraining the attention map in the decoder side. Our model, referred to as SMCA-RIPPLE, replaces all attention blocks with ripple attention in the transformer encoder. For completeness, we also compare with SMCA-LA, an SMCA variant that adopts linearized attention in encoder attention block.

Implementation details Note that RIPPLE is based on linearized attention mechanism (§2.2). In this work, the feature map $\phi(\cdot)$ is defined to be deterministic with learnable parameters, which consists of a two-layer MLP with trigonometric and ReLU activations in turn. We find it works very well in our experiments. Detailed discussions about our choice and a corresponding ablation study can be found in Appendix C.1.

Besides, in RIPPLE we introduce a hard constraint R_{max} such that the model stops rippling propagation at distance R_{max} and then merges all the remaining groups. This can be seen as a stronger version of halting threshold τ (§3.2) and is adopted in practice due to its explicit effect on controlling the rippling process.[8] Furthermore, when applied in vision transformers, we only replace the first several attention layers with ripple attention, while the remaining ones adopt linearized attention. We refer readers to Appendix C for more discussions about our implementation choices and further details.

4.2 MAIN RESULTS

Results on ImageNet-1K dataset. The results of comparisons among RIPPLE and other models on `ImageNet1k` dataset are presented in Table 1. We observe RIPPLE outperforms both DEIT-LA, upon which RIPPLE is built, and its variants by a large margin. Although DEIT-LA gives a clear

[7] To facilitate comparisons and simplify experimental settings, we do not use the distillation technique.

[8] Given R_{max}, our model is robust to the change of τ; therefore, we only set τ to 0.001 by default and mainly conduct ablation studies on R_{max} (§5.1).

Model	# Params	Top-1 Acc.	Top-5 Acc.
Models with quadratic complexity			
DEIT	5.72M	72.20	91.10
CONVIT (d'Ascoli et al., 2021)	5.72M	**73.11**	**91.71**
Models with sub-quadratic complexity			
DEIT-LA	5.76M	70.67	90.16
DEIT-LA + SINCSPE (Liutkus et al., 2021)	5.84M	67.32	88.14
DEIT-LA + CONVSPE (Liutkus et al., 2021)	6.69M	67.64	88.40
DEIT-LA + ROPE (Su et al., 2021)	5.76M	71.19	90.48
PERMUTEFORMER (Chen, 2021)	5.76M	71.42	90.51
RIPPLE	5.78M	73.02	91.56

Table 1: Image classification results for different vision transformers on `ImageNet1k` dataset. All the variants of DEIT-LA, including PERMUTEFORMER, are trained by us.

Model	w/ APE			w/o APE		
	# Params	Top-1 Acc.	Top-5 Acc.	# Params	Top-1 Acc.	Top-5 Acc.
DEIT-LA	5.42M	67.00	88.57	5.36M	54.04	79.66
DEIT	5.42M	67.87	89.71	5.36M	53.64	80.30
CONVIT	5.42M	**74.34**	**92.87**	5.36M	**73.88**	**92.20**
RIPPLE	5.47M	73.94	92.37	5.42M	72.94	91.86

Table 2: Image classification results for different vision transformers on `CIFAR-100` dataset. All of these models are trained by us. APE denotes the absolute positional embedding.

performance drop compared to the standard vision transformer DEIT, RIPPLE still performs better than DEIT and achieves results comparable to the improved variant CONVIT while in asymptotically faster runtime, which clearly demonstrates the effectiveness of our approach.

Results on CIFAR-100 dataset. We further conduct experiments on `CIFAR-100` dataset and report the results in Table 2. RIPPLE outperforms both DEIT-LA and DEIT by a substantial margin on `CIFAR-100` dataset, and also achieves competitive performance compared to CONVIT. This suggests that RIPPLE also generalizes well on a relatively smaller dataset. Following the setting in (Wu et al., 2021; Yan et al., 2021), we also make a comparison of these models in the absence of absolute positional embeddings. We observe a significantly larger performance gap between vanilla vision transformers and models designed to incorporate the notion of locality (Table 2). This implies RIPPLE could structure the scattered spatial context, which is beneficial to information aggregation among patches. Still, the performance decrease in RIPPLE in the absence of positional embeddings suggests that absolute global positions contain complementary information to the prior knowledge of locality, which is also consistent with a recent study (Islam et al., 2020).

Results on COCO benchmark. In Table 3 we report the results for object detection. Again, we see the same trend that the performance drops by over 2 AP when using linearized attention in the encoder (SMCA-LA). However, SMCA-RIPPLE improves SMCA-LA on all object scales with a marginal increase of GFLOPs and almost catches up with SMCA. The mAP gap between SMCA-RIPPLE and SMCA is further narrowed down from 0.5 to 0.3 with 108 training epochs. In addition, SMCA-RIPPLE achieves better results than SMCA on small scale objects, which is attributed to the promoted locality of ripple attention.

5 ANALYSIS

5.1 INSPECTING THE RIPPLING PROCESS

On the effect of maximum rippling distances. The maximum rippling distance R_{max} defined in §4.1 controls the boundary of RIPPLE with informative spatial weights. To evaluate its effect on the modeling performance, we vary the maximum rippling distance R_{max} and report the results on

Model	# Params	GFLOPs	Inference time(s)	50 epochs				108 epochs			
				AP	AP_S	AP_M	AP_L	AP	AP_S	AP_M	AP_L
SMCA	41.5M	88	0.059	**41.0**	21.9	**44.3**	**59.1**	**42.7**	22.8	**46.1**	**60.0**
SMCA-LA	41.7M	79	0.062	39.1	19.8	42.8	56.5	41.1	22.0	44.5	59.0
SMCA-RIPPLE	41.8M	80	0.065	40.5	**22.1**	44.1	57.7	42.3	**23.2**	45.6	**60.0**

Table 3: Object detection results for different detection transformers on COCO benchmark under both 50 training epoch schedule and 108 epoch schedule.

Model	locality	global dep.	R_{max}	Speed	Top-1 Acc.	Top-5 Acc.
RIPPLE	✓	✓	2	832	72.65	91.83
			4	792	**73.94**	**92.37**
			8	578	73.37	92.21
			16	382	73.48	92.25
FIXED-RIPPLE	✓	✓	4	795	71.34	90.77
TRUNCATED-RIPPLE	✓	✗	4	820	72.18	91.66
RIPPLE w/o SBT	✗	✓	4	784	71.94	91.83
DEIT-LA	✗	✓	–	2664	67.00	88.57

Table 4: Classification results on CIFAR-100 dataset under different maximum rippling distances R_{max} for RIPPLE, compared to several model variants. The speed is measured by the number of images processed per second with a batch size of 64. RIPPLE w/o SBT represents ripple attention whose spatial weights are generated through a softmax function instead of stick breaking transforms (SBT). "–" indicates not applicable.

CIFAR-100 dataset in Table 4. Overall, RIPPLE performs well with a moderate or larger R_{max}. If R_{max} is too small, the performance drops significantly, although still outperforming DEIT-LA. It can be attributed to the fact that if the stick-breaking transformation terminates early, the query would attend mostly to its immediate spatial neighbors while not sufficiently respecting global dependencies.

On the effect of stick-breaking transformation. We construct a variant of RIPPLE with fixed and exponentially-decayed spatial weights (i.e., $1/2, (1/2)^2, (1/2)^3, \cdots, (1/2)^{R_{\text{max}}}$), which is denoted by FIXED-RIPPLE. We find RIPPLE with hard-coded weights also performs better than DEIT-LA, which indicates the effectiveness of recovering spatial structures in transformers. Our stick-breaking transformation gives a further boost over the fixed-weight baseline, thanks to its flexibility in the spatial weights. We also consider a baseline RIPPLE w/o SBT, which replaces the stick-breaking transformation (§3.2) with a simple softmax function to generate spatial weights. This variant does not promote any locality but has the potential to learn such pattern implicitly. It performs slightly better than hard-coded spatial weights and much worse compared to RIPPLE, verifying the effectiveness of stick-breaking transformation.

On the effect of global and local information. To demonstrate the relation between global and local information, we design another baseline TRUNCATED-RIPPLE, which puts a hard termination of rippling process such that all distant groups beyond $R_{\text{max}} = 4$ are discarded (i.e., $\alpha_r(i,j) = 0$, if $r \geq 4$) instead of merged. This results in a limited receptive field without global dependency modeling. As shown in Table 4, the comparison among TRUNCATED-RIPPLE, RIPPLE and DEIT-LA reveals that both global and local information play an important role in modeling, while the notion of locality is possibly more important than global connectivity in vision tasks, which concurs with previous findings (Dosovitskiy et al., 2020; d'Ascoli et al., 2021).

5.2 Empirical running time and memory consumption

To verify the advantage of asymptotically faster running complexity in RIPPLE, we conduct a simulation experiment on vision transformers to compare the empirical running time and memory consumption of RIPPLE against its baselines under different numbers of tokens. The detailed setup can be found in Appendix E. Figures 2 demonstrate the comparison results. As mentioned in §3.3, both DEIT and RIPPLE (Naïve) come with quadratic complexity in the number of tokens. We observe that RIPPLE with dynamic programming (DP) performs significantly better than RIPPLE (Naïve), which demonstrates the effectiveness of our dynamic programming algorithm. Furthermore, RIPPLE

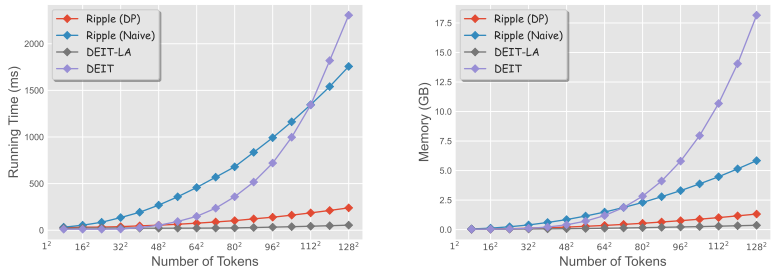


Figure 2: Empirical running time (left) and memory consumption (right) under different numbers of tokens. All models are tested with a batch size of 4 on a single NVIDIA V100 GPU machine, averaged by 10 runs.

behaves similarly to DEIT-LA as the number of tokens increases, verifying that it could be executed in linear observed time. When processing a large number of tokens, RIPPLE often achieves a $5\times$ or even $10\times$ reduction in running time and memory compared to its quadratic counterparts.

6 Related work

Transformer architectures (Vaswani et al., 2017) are first introduced for neural machine translation. Recently, researchers begin to apply the transformer model in the computer vision domain, showing promising results in various tasks, such as image generation (Parmar et al., 2018), video action recognition (Bertasius et al., 2021; Liu et al., 2021b), segmentation (Wang et al., 2020b; Strudel et al., 2021), object detection (Carion et al., 2020; Gao et al., 2021), low-level image processing (Chen et al., 2020) and image classification (Dosovitskiy et al., 2020; Touvron et al., 2020; Liu et al., 2021a). A large body of research has been devoted into improving efficiency and effectiveness of vision transformers (Dosovitskiy et al., 2020). Recent advances improve original vision transformers from various perspectives, such as data-efficient training (Touvron et al., 2020), adopting pyramid architectures (Wang et al., 2021a; Liu et al., 2021a; Heo et al., 2021) and incorporating the notion of locality, which can be done by applying convolutional modules into the architecture (Li et al., 2021; Wu et al., 2021; Yan et al., 2021; Xu et al., 2021; Yuan et al., 2021; d'Ascoli et al., 2021), restricting the scope of the self-attention (Liu et al., 2021a; Dong et al., 2021) or initializing self-attention maps as a convolution kernel (d'Ascoli et al., 2021). In contrast to these prior works, we directly model the locality inside the attention mechanism yet permit long-term dependencies, without relying on any convolutional operations or limiting the receptive field; at the same time, ripple attention runs in linear observed time so that the quadratic bottleneck in standard vision transformers can be greatly alleviated. Our work is orthogonal to previous works that modify the transformer architecture and it is worth exploring their combination to improve the overall vision transformer design.

Our model is built on the linearized attention mechanism, which approximates the softmax kernel with the dot product of feature maps. The feature maps can be stochastic, such as in RFA (Peng et al., 2021) and Performer (Choromanski et al., 2020), or deterministic (Katharopoulos et al., 2020; Schlag et al., 2021). Recently, many works are proposed to improve linearized attention by incorporating relative positional encodings (Liutkus et al., 2021; Luo et al., 2021; Chen, 2021; Su et al., 2021). Other efficient attention mechanisms include methods that limit the attention pattern to be sparse (Child et al., 2019; Ho et al., 2019; Kitaev et al., 2020) or utilizes a low rank approximation by projecting input sequences to fewer key-value pairs (Wang et al., 2020a). A comprehensive review of recent advances in efficient attention mechanisms can be found in Tay et al. (2020a;b).

7 Conclusion

In this work, we present ripple attention, a novel attention mechanism for visual perception with sub-quadratic complexity. In ripple attention, contributions of different tokens to a query are weighted with respect to their spatial distances in the 2D space. We design a dynamic programming algorithm that computes weighted contributions for all queries in linear observed time and derive the spatial weights through an adaptive stick-breaking transformation. We conduct extensive experiments and analyses to demonstrate the effectiveness of ripple attention.

REFERENCES

Alexei Baevski and Michael Auli. Adaptive input representations for neural language modeling. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ByxZX20qFQ.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Maxim Berman, Hervé Jégou, Andrea Vedaldi, Iasonas Kokkinos, and Matthijs Douze. Multigrain: a unified image embedding for classes and instances. *arXiv preprint arXiv:1902.05509*, 2019.

Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? In *Proceedings of the International Conference on Machine Learning (ICML)*, July 2021.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pp. 213–229. Springer, 2020.

Hanting Chen, Yunhe Wang, Tianyu Guo, Chang Xu, Yiping Deng, Zhenhua Liu, Siwei Ma, Chunjing Xu, Chao Xu, and Wen Gao. Pre-trained image processing transformer. *arXiv preprint arXiv:2012.00364*, 2020.

Peng Chen. Permuteformer: Efficient relative position encoding for long sequences. *arXiv preprint arXiv:2109.02377*, 2021.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Franklin C Crow. Summed-area tables for texture mapping. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pp. 207–212, 1984.

Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 702–703, 2020.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2978–2988, Florence, Italy, 2019. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P19-1285.

Stéphane d'Ascoli, Hugo Touvron, Matthew Leavitt, Ari Morcos, Giulio Biroli, and Levent Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. *arXiv preprint arXiv:2103.10697*, 2021.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

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.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Weiming Zhang, Nenghai Yu, Lu Yuan, Dong Chen, and Baining Guo. Cswin transformer: A general vision transformer backbone with cross-shaped windows. *arXiv preprint arXiv:2107.00652*, 2021.

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.

Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pp. 267–285. Springer, 1982.

Peng Gao, Minghang Zheng, Xiaogang Wang, Jifeng Dai, and Hongsheng Li. Fast convergence of detr with spatially modulated co-attention. *arXiv preprint arXiv:2101.07448*, 2021.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Byeongho Heo, Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Junsuk Choe, and Seong Joon Oh. Rethinking spatial dimensions of vision transformers. *arXiv preprint arXiv:2103.16302*, 2021.

Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*, 2019.

Elad Hoffer, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoefler, and Daniel Soudry. Augment your batch: Improving generalization through instance repetition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8129–8138, 2020.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pp. 646–661. Springer, 2016.

Md Amirul Islam, Sen Jia, and Neil D. B. Bruce. How much position information do convolutional neural networks encode? In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rJeB36NKvB`.

Jungo Kasai, Hao Peng, Yizhe Zhang, Dani Yogatama, Gabriel Ilharco, Nikolaos Pappas, Yi Mao, Weizhu Chen, and Noah A Smith. Finetuning pretrained transformers into rnns. *arXiv preprint arXiv:2103.13076*, 2021.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pp. 5156–5165. PMLR, 2020.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rkgNKkHtvB`.

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

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Yawei Li, Kai Zhang, Jiezhang Cao, Radu Timofte, and Luc Van Gool. Localvit: Bringing locality to vision transformers. *arXiv preprint arXiv:2104.05707*, 2021.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021a.

Ze Liu, Jia Ning, Yue Cao, Yixuan Wei, Zheng Zhang, Stephen Lin, and Han Hu. Video swin transformer. *arXiv preprint arXiv:2106.13230*, 2021b.

Antoine Liutkus, Ondřej Cífka, Shih-Lun Wu, Umut Simsekli, Yi-Hsuan Yang, and Gael Richard. Relative positional encoding for transformers with linear complexity. In *International Conference on Machine Learning*, pp. 7067–7079. PMLR, 2021.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Shengjie Luo, Shanda Li, Tianle Cai, Di He, Dinglan Peng, Shuxin Zheng, Guolin Ke, Liwei Wang, and Tie-Yan Liu. Stable, fast and accurate: Kernelized attention with relative positional encoding. *arXiv preprint arXiv:2106.12566*, 2021.

Depu Meng, Xiaokang Chen, Zejia Fan, Gang Zeng, Houqiang Li, Yuhui Yuan, Lei Sun, and Jingdong Wang. Conditional detr for fast training convergence. *arXiv preprint arXiv:2108.06152*, 2021.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International Conference on Machine Learning*, pp. 4055–4064. PMLR, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32: 8026–8037, 2019.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A Smith, and Lingpeng Kong. Random feature attention. *arXiv preprint arXiv:2103.02143*, 2021.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International Conference on Machine Learning*, pp. 9355–9366. PMLR, 2021.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *NAACL-HLT (2)*, 2018.

Eero P Simoncelli and Bruno A Olshausen. Natural image statistics and neural representation. *Annual review of neuroscience*, 24(1):1193–1216, 2001.

Robin Strudel, Ricardo Garcia, Ivan Laptev, and Cordelia Schmid. Segmenter: Transformer for semantic segmentation. *arXiv preprint arXiv:2105.05633*, 2021.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020a.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*, 2020b.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. *arXiv preprint arXiv:2012.12877*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Paul Viola, Michael Jones, et al. Robust real-time object detection. *International journal of computer vision*, 4(34-47):4, 2001.

Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020a.

Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *arXiv preprint arXiv:2102.12122*, 2021a.

Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pvtv2: Improved baselines with pyramid vision transformer. *arXiv preprint arXiv:2106.13797*, 2021b.

Yuqing Wang, Zhaoliang Xu, Xinlong Wang, Chunhua Shen, Baoshan Cheng, Hao Shen, and Huaxia Xia. End-to-end video instance segmentation with transformers. *arXiv preprint arXiv:2011.14503*, 2020b.

Larry Wasserman. *All of nonparametric statistics*. Springer Science & Business Media, 2006.

Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. *arXiv preprint arXiv:2103.15808*, 2021.

Tete Xiao, Mannat Singh, Eric Mintun, Trevor Darrell, Piotr Dollár, and Ross Girshick. Early convolutions help transformers see better. *arXiv preprint arXiv:2106.14881*, 2021.

Yufei Xu, Qiming Zhang, Jing Zhang, and Dacheng Tao. Vitae: Vision transformer advanced by exploring intrinsic inductive bias. *arXiv preprint arXiv:2106.03348*, 2021.

Haotian Yan, Zhe Li, Weijian Li, Changhu Wang, Ming Wu, and Chuang Zhang. Contnet: Why not use convolution and transformer at the same time? *arXiv preprint arXiv:2104.13497*, 2021.

Jianwei Yang, Chunyuan Li, Pengchuan Zhang, Xiyang Dai, Bin Xiao, Lu Yuan, and Jianfeng Gao. Focal self-attention for local-global interactions in vision transformers. *arXiv preprint arXiv:2107.00641*, 2021.

Kun Yuan, Shaopeng Guo, Ziwei Liu, Aojun Zhou, Fengwei Yu, and Wei Wu. Incorporating convolution designs into visual transformers. *arXiv preprint arXiv:2103.11816*, 2021.

Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6023–6032, 2019.

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 13001–13008, 2020.

Appendices

A Analysis of runtime complexity

In this section, we give more details about the runtime complexity (§3.3) for different attention mechanisms in Table 5. In particular, we focus on variants (1) Ripple-softmax and (2) Ripple (naïve), since the rest have been clarified in the main text.

Complexity of ripple-softmax. Ripple-softmax aims to inject radial bias through rippling into the vanilla softmax attention. Here we present two possible ways to achieve this goal, either explicitly or implicitly. Given a query, we could perform vanilla attention over one of its vicinal groups once so that the rippling effect could be explicitly encoded. Since the number of tokens within vicinal group of distance r is directly proportional to r (note that the number of tokens within vicinal group of distance r is simply the difference between $(2r+1)^2 - (2r-1)^2 = 8r$) and there are R groups in total, the overall complexity would be $\mathcal{O}(R^2)$; then considering all queries results in $\mathcal{O}(HWR^2)$ complexity (along with a **large** constant). On the other hand, we could also add specific spatial weights directly to the attention matrix, implicitly enforcing the radial bias. This is similar to relative positional encodings (Shaw et al., 2018), but in this case the overall complexity is at least $\mathcal{O}(H^2W^2)$ due to the computation of attention matrices, which is as inefficient as vanilla softmax attention in terms of complexity. In this work, we simply refer to the explicit method as Ripple-softmax.

Complexity of Ripple (naïve). Naïvely implementing ripple attention comes with $\mathcal{O}(HWR^2)$ complexity, as mentioned in the beginning of § 3.3. This is due to the fact that for each query, we have to first sum over all tokens in one vicinal group, which is $\mathcal{O}(r)$ (see the analysis in the paragraph above), and then aggregate over all groups. This results in $\mathcal{O}(R^2)$ complexity, and $\mathcal{O}(HWR^2)$ for all queries (also with a large constant; see the empirical running comparison in §5.2).

B Efficient gradient computation in ripple attention

To perform gradient back-propagation for ripple attention, a naive implementation would be directly adopting the automatic differentiation, since all operations in our forward pass (Algorithm 1) are differentiable; however, since many computations in Algorithm 1 overlap with each other, it would lead to substantially repetitive calculations. In addition, we find it even takes $\mathcal{O}(T^2)$ time and memory, which is highly inefficient compared to the forward pass.

In this section, we present an algorithm based on dynamic programming to perform efficient back-propagation for ripple attention, which again comes with sub-quadratic complexity. Recall that given a single query $\boldsymbol{q}_{ij}$, all the keys $\boldsymbol{K}$ and values $\boldsymbol{V}$, ripple attention executes the following computation

Softmax attention	Ripple-softmax†	Linearized attention	Ripple (naïve)	**Ripple (DP)**
$\mathcal{O}\left(H^2W^2\right)$	$\mathcal{O}\left(HWR^2\right)$	$\mathcal{O}\left(HW\right)$	$\mathcal{O}\left(HWR^2\right)$	$\mathcal{O}\left(HWR\right)$
$\mathcal{O}\left(T^2\right)$	$\mathcal{O}\left(T^2\right)$	$\mathcal{O}\left(T\right)$	$\mathcal{O}\left(T^2\right)$	$\mathcal{O}\left(T^{3/2}\right)$

Table 5: Runtime complexity comparisons between different attention variants, with respect to an image with $H \times W$ patches (in the first row) and with respect to the number of patch tokens $T \coloneqq H \times W$ (in the second row). †Ripple-softmax indicates the complexity if we would like to implement ripple-like mechanisms in vanilla softmax attention.

during the forward pass[9]:

$$\text{RippleAttn}\left(\boldsymbol{q}_{ij}, \boldsymbol{K}, \boldsymbol{V}\right) \coloneqq \frac{\boldsymbol{q}_{ij}^\top \sum_{r=0}^{R} \alpha_r(i,j) \sum_{(m,n)\in\mathcal{N}_r(i,j)} \boldsymbol{k}_{mn}\boldsymbol{v}_{mn}^\top}{\boldsymbol{q}_{ij}^\top \sum_{r=0}^{R} \alpha_r(i,j) \sum_{(m',n')\in\mathcal{N}_r(i,j)} \boldsymbol{k}_{m'n'}}. \tag{10}$$

The main difference between linearized attention and ripple attention lies in the computation procedure of summation over $\boldsymbol{k}\boldsymbol{v}^\top$ and $\boldsymbol{k}$. Therefore, we put main focus on calculating gradients for the following quantity

$$\boldsymbol{y}_{ij} \coloneqq \sum_{r=0}^{R} \alpha_r(i,j) \sum\nolimits_{(m,n)\in\mathcal{N}_r(i,j)} \boldsymbol{x}_{mn}, \tag{11}$$

where $\boldsymbol{x}_{mn} \in \mathbb{R}^D$ denotes a D-dimensional vector located at position (m,n), which could be (unrolled) $\boldsymbol{k}_{mn}\boldsymbol{v}_{mn}^\top$ or $\boldsymbol{k}_{mn}$.[10] The remaining computation in ripple attention (e.g., dot product with $\boldsymbol{q}_{ij}$) can be easily handled by standard automatic differentiation. We are mainly interested in computing gradients with respect to $\alpha_r(i,j)$ and $\boldsymbol{x}_{mn}$

In ripple attention, we maintain a summed area table (SAT) to efficiently retrieve the *partial* reduction over vicinal groups (see Algorithm 1 for more details). Although all operations in Algorithm 1 is differentiable and thus admits the use of automatic differentiation to calculate gradients, it is very inefficient since computations of most intermediate nodes in the computation graph (for example, $\boldsymbol{S}_1$, $\boldsymbol{S}_2$, $\boldsymbol{S}_3$ and $\boldsymbol{S}_4$ in Algorithm 1) overlap with each other, resulting in a large amount of repeated computation.

Here we inspect the form (equation 11) and show that the properties of vicinal groups could be made use of to derive efficient gradient computation.

Gradients with respect to spatial weights. We assume gradients $\nabla_{\boldsymbol{y}_{ij}}\mathcal{L}$, that is, the gradient of our loss objective w.r.t. output at all positions (i,j) are available during back-propagation. According to the chain rule, the partial derivative of the objective $\mathcal{L}$ w.r.t. $\alpha_r(i,j)$ has the following form:

$$\frac{\partial\mathcal{L}}{\partial\alpha_r(i,j)} = \sum_{d=1}^{D} \frac{\partial\mathcal{L}}{\partial y_{ijd}} \frac{\partial y_{ijd}}{\partial\alpha_r(i,j)} = \sum_{d=1}^{D} \frac{\partial\mathcal{L}}{\partial y_{ijd}} \sum\nolimits_{(m,n)\in\mathcal{N}_r(i,j)} x_{mnd},$$

where x_{mnd} and y_{mnd} denote the d-th dimension of $\boldsymbol{x}_{mn}$ and the output $\boldsymbol{y}_{mn}$ respectively. The first quality holds since spatial weights at every position only depends on the output at that position; but since the same spatial weight applies to all dimensions of $\boldsymbol{y}_{ij}$, we have to reduce over the embedding dimension to compute the partial derivative. Similar to the forward pass computation (equation 9), we recognize that the inner summation over the vicinal group $\mathcal{N}_r(i,j)$ can be again computed efficiently by utilizing SATs with Algorithm 1.

Gradients with respect to $\boldsymbol{x}_{mn}$. The partial derivative w.r.t. element x_{mnd} can be written as

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial x_{mnd}} &= \sum_{i=1}^{H}\sum_{j=1}^{W} \frac{\partial\mathcal{L}}{\partial y_{ijd}} \frac{\partial y_{ijd}}{\partial x_{mnd}} \\ &= \sum_{i=1}^{H}\sum_{j=1}^{W} \frac{\partial\mathcal{L}}{\partial y_{ijd}} \sum_{r=0}^{R} \alpha_r(i,j) \mathbb{I}\left[(m,n)\in\mathcal{N}_r(i,j)\right].\end{aligned} \tag{12}$$

where we define $\mathbb{I}\left[(m,n)\in\mathcal{N}_r(i,j)\right]$ as the indicator function such that it is set to 1 if $(m,n)\in\mathcal{N}_r(i,j)$ and 0 otherwise. A naive way to compute the partial derivatives above has $\mathcal{O}(H^2W^2R)$ complexity, since for every key vector at position (m,n) we need to sum its influences over all positions. However, we show that we could again solve them via dynamic programming.

[9]Without loss of generality, we merge the feature map $\phi(\cdot)$ into the vector representation of queries and keys to simplify notations.

[10]We focus on the general form here since the derived algorithm applies to both the nominator and the denominator.

Our key observation is that the vicinal group is symmetrical w.r.t. its arguments, that is, $(m, n) \in \mathcal{N}_r(i, j)$ if and only if $(i, j) \in \mathcal{N}_r(m, n)$. Then the partial derivative (equation 12) is equivalent to

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial x_{mnd}} &= \sum_{i=1}^{H}\sum_{j=1}^{W} \frac{\partial \mathcal{L}}{\partial y_{ijd}} \sum_{r=0}^{R} \alpha_r(i,j) \mathbb{I}\left[(m,n) \in \mathcal{N}_r(i,j)\right] \\
&= \sum_{i=1}^{H}\sum_{j=1}^{W} \frac{\partial \mathcal{L}}{\partial y_{ijd}} \sum_{r=0}^{R} \alpha_r(i,j) \mathbb{I}\left[(i,j) \in \mathcal{N}_r(m,n)\right] \\
&= \sum_{r=0}^{R}\sum_{i=1}^{H}\sum_{j=1}^{W} \mathbb{I}\left[(i,j) \in \mathcal{N}_r(m,n)\right] \frac{\partial \mathcal{L}}{\partial y_{ijd}} \alpha_r(i,j) \\
&= \sum_{r=0}^{R} \sum\nolimits_{(i,j)\in\mathcal{N}_r(m,n)} \frac{\partial \mathcal{L}}{\partial y_{ijd}} \alpha_r(i,j).
\end{aligned}$$

Thanks to the symmetry, the computation of partial derivatives is converted into reduction over vicinal groups, it can be effectively solved by dynamic programming (§3.3) in again $\mathcal{O}(HWR)$ time, which involves instantiating an SAT for the quantity $\frac{\partial \mathcal{L}}{\partial y_{ijcd}} \alpha_r(i, j)$ over all positions (i, j). Equipped with this result, substituting $\boldsymbol{k}_{mn}\boldsymbol{v}_{mn}^\top$ or $\boldsymbol{k}_{mn}$ into $\boldsymbol{x}_{mn}$ yields the term in the nominator and denominator, respectively.

C Additional implementation details

We implement our model using PyTorch (Paszke et al., 2019) and PyTorch image models (`timm`) toolkit (Wightman, 2019). We also implement a CUDA kernel for the ripple attention mechanism.

C.1 Deterministic adaptive feature maps for linearized attention

Background. Generally, a random feature map $\phi_\omega(\cdot)$ is defined by a function $h(\cdot) : \mathbb{R}^D \to \mathbb{R}$, m uni-variate functions $f_1, f_2, \ldots, f_m : \mathbb{R} \to \mathbb{R}$ as well as d identically distributed random vectors $\omega_1, \omega_2, \ldots, \omega_d$ following some distribution (Choromanski et al., 2020):

$$\phi_\omega(\boldsymbol{x}) \coloneqq \frac{h(x)}{\sqrt{d}} \left[f_1(\omega_1^\top \boldsymbol{x}), \ldots, f_1(\omega_d^\top \boldsymbol{x}), \ldots, f_m(\omega_1^\top \boldsymbol{x}), \ldots, f_m(\omega_d^\top \boldsymbol{x})\right]$$

yielding a map from $\mathbb{R}^D$ to $\mathbb{R}^{D'}$, where $D' = md$. Then by setting different configurations of f's, ω's and h, we could construct various unbiased estimators for the quantity $\exp(\boldsymbol{x}^\top \boldsymbol{y})$, that is,

$$\exp(\boldsymbol{x}^\top \boldsymbol{y}) = \mathbb{E}_{\omega_1,\ldots,\omega_d}\left[\phi_\omega(\boldsymbol{x})^\top \phi_\omega(\boldsymbol{y})\right]$$

For instance, we could let $m = 2$, where $f_1 = \sin$, $f_2 = \cos$ are trigonometric functions and $h(\boldsymbol{x}) = \exp\left(||x||^2/2\right)$ (Peng et al., 2021; Choromanski et al., 2020). Although unbiased, researchers note that the use of trigonometric functions does not ensure non-negative scores, which may lead to large estimate variance and unstable training (Choromanski et al., 2020). Alternatively, we could construct an estimator by setting $m = 1$ with $f_1 = \exp$ and $h(\boldsymbol{x}) = \exp\left(-||x||^2/2\right)$, which is again unbiased but enjoys positiveness (FAVOR+, Choromanski et al., 2020).

Our proposed deterministic adaptive feature map. Recently, researchers also proposed various heuristic designs of feature maps (Choromanski et al., 2020; Schlag et al., 2021; Kasai et al., 2021) that do not guarantee unbiasedness but might either exhibit lower variance, simplify computation or bring other useful benefits. Unfortunately, through extensive preliminary experiments we found most of these linearized attention variants (either random or deterministic) did not work well in the setting of vision transformers. We hypothesize there are two reasons for the performance drop: the first one is the usage of random samples, which suffers from the slow Monte Carlo convergence rate and instability during training; the second one is due to fixed weights, preventing the map from being adaptive and learning useful patterns. To this end, we propose the following deterministic feature map:

$$\phi(\boldsymbol{x}) = \mathrm{ReLU}(\boldsymbol{W}_2[\sin(\boldsymbol{W}_1\boldsymbol{x}); \cos(\boldsymbol{W}_1\boldsymbol{x})] + \boldsymbol{b}_2). \tag{13}$$

Feature map	Deterministic	Top-1 Acc.
RFA (Peng et al., 2021)	✗	67.10
Performer (Choromanski et al., 2020)	✗	65.92
DPFP (Schlag et al., 2021)	✓	63.95*
T2R (Kasai et al., 2021)	✓	70.02
Ours	✓	**70.67**
Ours w/ randomly sampled $\boldsymbol{W}_1$	✗	66.82
Ours w/o fully connected network	✓	70.02†

Table 6: Classification results on `ImageNet1k` dataset under different choices of feature maps. † indicates that our feature map design without fully connected network is identical to T2R (Kasai et al., 2021). * denotes the model does not fully converge.

Intuitively, we still follow the trigonometric feature map, except that we set $\boldsymbol{W}_1$ to be initialized as independent standard Gaussian samples but then learnable during training; the generated feature is then passed through a fully connected layer followed by a ReLU activation. It is deterministic and involves learnable parameters, which we found greatly improves performance.

Comparison with other feature maps and ablation study. We conduct a simple ablation study to demonstrate the effectiveness of our proposed feature map and report comparisons with other feature maps[11], as shown in Table 6. In general, we find it works pretty well in practice and outperforms other feature maps that are either deterministic or random. For our ablation study, we consider two variants of our proposed approach: (1) the method that recovers the original random trigonometric feature map, that is, recasting $\boldsymbol{W}_1$ as random samples and re-drawing it at every iteration; (2) the method that removes the fully connected layer (characterized by parameters $\boldsymbol{W}_2$ and $\boldsymbol{b}_2$). From Table 6, we see a great performance drop if we use random weights, which indicates that random feature maps lead to more difficult training in vision transformers. In addition, a feed-forward layer will give a further performance boost due to the increased flexibility. Therefore, we adopt our proposed deterministic feature map throughout our work.

C.2 Explicitly controlling the maximum rippling distance.

In §3.2 we define the threshold τ to control the termination of rippling process. In practice we find it beneficial to introduce a hard constraint R_{max} such that the model limits the maximum distance of rippling propagation to R_{max} and then merges all the remaining groups. In this way, we could not only further reduce the computation overhead, but also encourage the attention mechanism to allocate more weights to distal groups. This can be seen as a stronger version of halting threshold τ, which is easier to tune due to a more intuitive effect on the rippling process. We find an intermediate value gives a reasonable trade-off between local and long-term dependencies. Given R_{max}, our model is robust to the change of τ; therefore, we only set τ to 0.001 by default and mainly conduct ablation studies on R_{max}.

C.3 The parameterization of spatial weights

In terms of parameterizing spatial weights, we allocate an embedding vector for every of R_{max} stick units, so that they could adapt themselves to learn useful patterns from data. To compute spatial weights, we first linearly project each value vector $\boldsymbol{v}_{ij}$ and then perform dot-product with each of R_{max} stick unit embeddings[12] to produce R_{max} logits $\{\mathrm{o}_r(i,j)\}_{r=1}^{R_{\text{max}}}$. Every logit is then passed through a modified sigmoid function to yield the length of each stick unit $\mathrm{s}_r(i,j) =$

[11] For methods that adopt random features, we sample a set of random weights at every training step and use the same set of weights during evaluation. We also attempted various ways to schedule the redrawing random weights during training, but did not observe any performance gain.

[12] Since stick-breaking transformations ensure the produced weights to be inside a simplex, R_{max} logits would suffice to produce a sequence of spatial weights with size $R_{\text{max}} + 1$.

$1/\left[1+(R_{\text{max}}-r)\exp\left(-\text{o}_r(i,j)\right)\right]$. This modification, which is inspired by the default stick-breaking transform implementation in PyTorch distribution package (Paszke et al., 2019), ensures the model does not put most of mass on the first several sticks. We find this trick slightly improves performance. Consequently, spatial weights $\{\alpha_r(i,j)\}_{r=0}^{R_{\text{max}}}$ are derived by applying stick-breaking transformations to $\text{s}_r(i,j)$'s according to equation 5.

C.4 ARCHITECTURE DETAILS

For image classification, all model architectures follow the tiny variant of DEIT (Touvron et al., 2020), consists of 12 transformer layers, with the embedding dimension set to 192, except that we set the number of heads per attention block to 6 for all models. For object detection, our model is based on the architecture of SMCA with single scale features (Gao et al., 2021), which could facilitate comparisons and demonstrate the effectiveness of ripple attention more clearly. In particular, the number of transformer layers is 6 for both the encoder and decoder, with the number attention heads and the embedding dimension set to 8 and 256, respectively; the backbone is the pre-trained ResNet-50 (He et al., 2016) on `ImageNet1k` with fixed batch-norm layers.

C.5 SPECIFICS FOR APPLYING RIPPLE ATTENTION IN VISION TRANSFORMERS

Average pooling instead of using class tokens for classification. Since ripple attention directly operates on 2D images, it is hard to directly employ the widely used class token for classification tasks (Dosovitskiy et al., 2020; Touvron et al., 2020). Instead, we adopt mean average pooling over all tokens instead of class tokens to extract feature vectors that are fed into the classification head.

Multi-head ripple attention. Similar to multi-head attention (Vaswani et al., 2017), which is used in most Vision transformer architectures, we also adopt a multi-head variant of ripple attention, where different heads maintain different sets of spatial weights. The multi-head ripple attention allows different heads to focus on locality to various degrees, increasing the overall expressiveness.

On the number of ripple layers. A straightforward implementation choice is to replace regular attention at all layers of ViT with ripple attention. However, we argue this is sub-optimal. Since the input tokens of transformers consist of local patches, promoting local correlations at lower layers and maintaining structural spatial contexts could facilitate information aggregation; but as tokens go higher, every token is contextualized by global information and in this case adding the notion of locality might mislead the modeling. Therefore, we propose to use a hybrid architecture, where the lower layers use ripple attention while upper ones still adopt linear attention mechanisms. This choice is further supported by our ablation study experiments Appendix D.1, where our model achieves the best performance over various settings if only the first 9 transformer layers use ripple attention. Therefore, throughout experiments we use this configuration unless otherwise stated.

C.6 TRAINING SETUP

In this section, we describe our full training setup for both image classification and object detection.

Training details for image classification We following the same procedure to train the models as in DEIT (Touvron et al., 2020), including the data-augmentation, the regularization and the hyper-parameter setting for a head-to-head comparison. We use AdamW optimizer (Loshchilov & Hutter, 2019) to train our model on 8 NVIDIA V100 GPUs for 300 epochs on both `CIFAR-100` and `ImageNet1k` datasets. We adopt commonly used data augmentation methods, including random clipping, cropping, Rand-Augment (Cubuk et al., 2020) and random erasing (Zhong et al., 2020). However, we remove repeated augmentation (Hoffer et al., 2020) as we find it slows down convergence for both linearized attention and ripple attention, as also observed in previous studies (Berman et al., 2019; Xiao et al., 2021). For regularization, we employ stochastic depth (Huang et al., 2016), Mixup (Zhang et al., 2017), Cutmix (Yun et al., 2019), all of which are set to default settings in DeiT (Touvron et al., 2020). Training protocols that are specific to different datasets are listed as follows:

# ripple layers	speed	Top-1 Acc.	Top-5 Acc.
0	2664	70.67	90.16
3	1355	71.63	90.42
6	916	72.41	90.32
9	792	**73.02**	**91.56**
12	563	72.69	91.30

Table 7: Classification results on `ImageNet1k` dataset under different numbers of rippling layers for RIPPLE. The speed is measured by the number of images processed per second with a batch size of 64 on a single NVIDIA V100 GPU machine, averaged by 5 runs.

- For `ImageNet1k` dataset we set the batch size to 1024 and the learning rate to 0.001 with cosine learning rate decay (Loshchilov & Hutter, 2016). The image size is set to 224×224 with patch size 16, resulting in 14×14 tokens.
- for `CIFAR-100` dataset, the batch size and the learning rate is set to 512 and 0.0005 respectively, with the same cosine learning rate decay. In terms of the image size, we use the original scale 32×32, where a patch size 2 is used to produce 16×16 non-overlapping patches.

During evaluation, we report top-1 and top-5 accuracy on the evaluation set of both `ImageNet1k` and `CIFAR-100` datasets.

Training details for object detection We follow the same training protocol as SMCA (Gao et al., 2021). In particular, we initialize the transformer parameters with Xavier initialization (Glorot & Bengio, 2010), and use the pretrained weights on `ImageNet1k` for the backbone. We adopt the AdamW optimizer (Loshchilov & Hutter, 2019), set the weight decay to 10^{-4} and the learning rate to 10^{-5} and 10^{-4} for the backbone and transformer, respectively. We also decrease the learning rate to 1/10 of its original value after 40 epochs for 50 epoch schedule and after 80 epochs for 108 epoch training schedule. The dropout rate is set to 0.1. The data augmentation scheme and the loss objective is also the same as SMCA (Gao et al., 2021). All detection models are trained on 8 NVIDIA V100 GPUs with a total batch size of 16.

D ADDITIONAL EXPERIMENT RESULTS

D.1 ON THE EFFECT OF VARIOUS RIPPLE LAYERS

As mentioned in §4.1, directly replacing all attention layers in DEIT-LA with RIPPLE could be a sub-optimal choice. To validate this, we conduct an ablation study on `ImageNet1k` dataset to investigate the effect of different numbers of ripple layers, where the first several layers use ripple attention while upper ones still adopt linearized attention mechanism. The results are shown in Table 7. In particular, we find the model performance consistently improves as the number of ripple layers increases, but drops a little when the depth of ripple layers reaches a certain level (e.g., 9). Our observation aligns with our intuition, which suggests using hybrid attention layers could achieve a good trade-off between locality promotion and global dependency modeling. Therefore, RIPPLE uses 9 ripple layers by default throughout our experiments unless otherwise stated.

D.2 ON THE EFFECT OF DIFFERENT PARAMETERIZATION SCHEMES FOR SPATIAL WEIGHTS

Ripple attention is a flexible framework in that it allows the trade-off between the running time complexity and task accuracy. To explore this, we compare the full ripple attention against a rippling process where ripples get exponentially thicker so that the process could reach the image boundary in logarithmic time. This model, which we refer to as RIPPLE-LOGARITHMIC, enjoys $\mathcal{O}(T \log T)$ time complexity and is more efficient than base ripple attention. To see this, we plot the empirical running statistics of RIPPLE-LOGARITHMIC under different numbers of tokens. For completeness, we also include a variant where R_{max} scales linearly with the image height (or width), denoted by RIPPLE-DENSE. As shown in Figure 3, we observe RIPPLE-LOGARITHMIC runs as fast as base ripple (whose R_{max} is fixed) and becomes more efficient than the dense version as the number of tokens

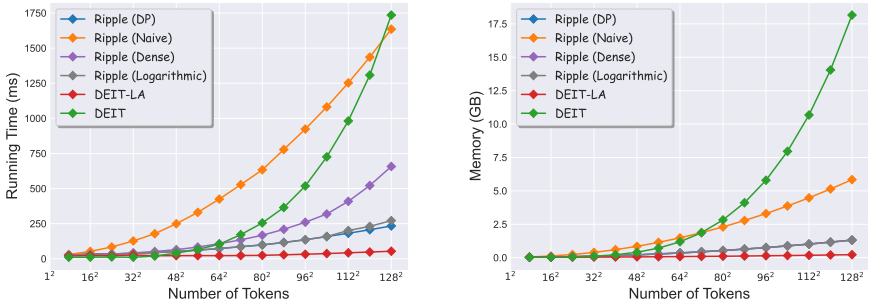


Figure 3: Empirical running time (left) and memory consumption (right) under different numbers of tokens, averaged by 5 runs.

increases. On the other hand, all of these models run with the same amount of memory consumption, as their space complexity is constant in the rippling distance. In terms of task performance, as reported in Table 8, we see a clear performance drop if we adopt RIPPLE-LOGARITHMIC, which could be due to that RIPPLE-LOGARITHMIC processes visual tokens at a coarser-grained level. This again justifies the flexibility of our framework: one could trade off the task accuracy for more efficiency and vice versa.

D.3 On the training dynamic of spatial weights

Recall that in §5.1 we compare RIPPLE against a baseline with fixed exponentially decayed spatial weights (denoted by FIXED-RIPPLE), which appears to be a trivial solution for incorporating the locality. However, it does not respect potentially strong long-term dependencies and only assign diminishing weights to distal groups. To further investigate how spatial weights generated from our proposed stick-breaking transform (SBT;§3.2) differ from FIXED-RIPPLE, we plot the training dynamic of the average[13] Jensen-Shannon divergence (JSD) between distributions induced by SBT and FIXED-RIPPLE, which is shown in Figure 4. Intuitively, a higher JSD value reflects a large discrepancy between their induced distributions. Since the logits in SBT are usually initialized around 0, the spatial weights are close to the exponential weights at first; however, as soon as the training starts, the JSD value rises sharply, which is possibly due to balancing between global and local information; after that, the curve decreases slightly, indicating that the mechanism might tend to favor local correlations; finally it plateaus at a high JSD value, which indicates that the induced distribution does not simply degenerate to a fixed distribution nor to a vanilla linearized attention (with uniform weights).

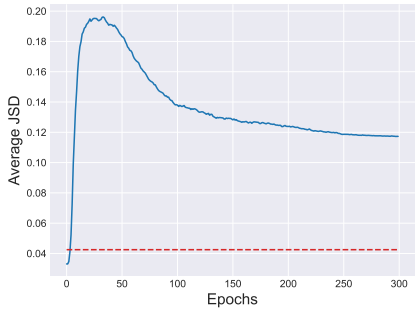


Figure 4: Training dynamic of average JSD between induced distributions of the proposed stick-breaking transform and fixed exponentially decayed weights. The solid line denotes the training dynamic of JSD between SBT and FIXED-RIPPLE, while the dashed line denotes the JSD between the uniform distribution (as in vanilla linearized attention) and FIXED-RIPPLE.

Scheme	Top-1 Acc.	Top-5 Acc.
Logarithmic	73.26	91.98
Dense	74.43	92.37

Table 8: Classification results on CIFAR-100 dataset under different parameterization schemes of spatial weights.

[13] We average the computed JSD score over all training samples, for each of which we further average over all attention blocks and heads.

E SETUP FOR EMPIRICAL RUNNING TIME AND MEMORY CONSUMPTION.

For the simulation experiment conducted in §5.2, we use the same vision transformer architecture for all models, whose hyper-parameter setting is specified in Appendix C.4, except that we set embedding dimension to 96 and batch size to 4; otherwise, most configurations tested here will make it infeasible for DEIT and RIPPLE (Naïve) to fit into the 32GB memory of a single NVIDIA V100 GPU machine.