Supplementary materials for Multi-Scale Vision Longformer: A New Vision Transformer for High-Resolution Image Encoding

A. Settings

A.1. Model configurations

We listed the model configuration of all models used in this paper in Table 8. We do not specify the attention mechanism here, because the model configuration is the same for all attention mechanisms and the attention-specific parameters are specified in Table 15.

Model	Stage1 n,p,h,d	Stage2 n,p,h,d	Stage3 n,p,h,d	Stage4 n,p,h,d
Tiny	1,4,1,48	1,2,3,96	9,2,3,192	1,2,6,384
Small	1,4,3,96	2,2,3,192	8,2,6,384	1,2,12,768
Medium	1,4,3,96	4,2,3,192	16,2,6,384	1,2,12,768
Base	1,4,3,96	8,2,3,192	24,2,6,384	1,2,12,768
Tiny 1-10-1	1,8,3,96		10,2,3,192	1,2,6,384
Tiny 2-9-1	2,8,3,96		9,2,3,192	1,2,6,384
Tiny 1-9-2	1,8,3,96		9,2,3,192	2,2,6,384
Tiny 2-8-2	2,8,3,96		8,2,3,192	2,2,6,384
Tiny 1-1-9-1	1,4,1,48	1,2,3,96	9,2,3,192	1,2,6,384
Tiny 1-2-8-1	1,4,1,48	2,2,3,96	8,2,3,192	1,2,6,384
Small 1-10-1	1,8,3,192		10,2,6,384	1,2,12,768
Small 2-9-1	2,8,3,192		9,2,6,384	1,2,12,768
Small 1-9-2	1,8,3,192		9,2,6,384	2,2,12,768
Small 2-8-2	2,8,3,192		8,2,6,384	2,2,12,768
Small 1-1-9-1	1,4,3,96	1,2,3,192	9,2,6,384	1,2,12,768
Small 1-2-8-1	1,4,3,96	2,2,3,192	8,2,6,384	1,2,12,768

Table 8. Model architecture for multi-scale stacked ViTs. Architecture parameters for each E-ViT module E-ViT$(a \times n/p\,; h, d)$: number of attention blocks n, input patch size p, number of heads h and hidden dimension d. See the meaning of these parameters in Figure 1 (Bottom).

A.2. Experimental settings

Table 9 summarizes our training setups for our different models.

For the ImageNet classification task, our setting mainly follow that in DeiT [43]. For example, we do not use dropout but use random path. We use all data augmentations in DeiT [43], except that we apply Repeated Augmentation only on Medium and Base models. When fine-tuning from a ImageNet-21K pretrained checkpoint, we mainly follow the practice of ViT [12], train on image size 384×384, use SGD with momentum 0.9, use no weight decay, and use only random cropping for data augmentation.

For COCO object detection/segmentation tasks, we follow the standard "$1\times$" and "$3 \times +$MS" schedules. We only change the optimizer from SGD to AdamW and search for good initial learning rate and weight decay. For the "$1\times$" schedule, the input image scale is fixed to be $(800, 1333)$ for the min and max sizes, respectively. For the "$3 \times +$MS" schedule, the input image is randomly resized to have min size in $\{640, 672, 704, 736, 768, 800\}$. We found that there is obvious over-fitting in Training ViL-Medium and ViL-Base models on COCO, mainly because that these two models are relatively large but they are only pretrained on ImageNet. Therefore, we are taking the best checkpoint (one epoch per checkpoint) along the training trajectory to report the performance.

B. More experimental results

B.1. Ablation study on the architecture design of multi-scale Vision Longformer

In this section, we present two ablation studies on the model architecture of multi-scale Vision Longformer.

Ablation of the effects of LayerNorm and 2-D positional embedding in the patch embedding. In Table 2, we show that our flat model E-ViT(full $\times$ $12/16$), which only differs from the standard ViT/DeiT model by an newly-added LayerNorm after the patch embedding and the 2-D positional embedding, has better performance than the standard ViT/DeiT model. In Table 10, we show that this better performance comes from the newly-added LayerNorm.

Feature from the CLS token or from average pooling? As shown in Table 11, for ViL models that has only one attention block in the last stage (ViL 1-2-8-1), the average pooled feature from all tokens works better than the feature of the CLS token. However, when there are more than 2 attention blocks in the last stage (ViL 1-1-8-2), the difference between these two features disappears. The ViL 1-1-8-2 model has better performance than the ViL 1-2-8-1 model because it has more trainable parameters.

B.2. A comprehensive comparison of different attention mechanisms on ImageNet classification

We compare different attention mechanisms with different model sizes and architectures in Table 12 and Table 13. In Table 12, we show their performance on ImageNet-1K classification problem, measured by Top-1 accuracy. In Table 13, we show their number of parameters and FLOPs. We would like to comment that FLOPs is just a theoretical estimation of computation complexity, and it may not fit well the space/time cost in practice.

B.3. Object detection and instance segmentation with 1x schedule

We report the results for the standard 1× schedule in Table 14, for both the RetinaNet and the Mask R-CNN

Model	Dataset	Epoch	Base Lr	LR decay	Weight decay	Drop Path	Batch size
MsViT-Tiny	ImageNet	300	1e-3	cosine	0.1	0.1	1024
MsViT-Small	ImageNet	300	1e-3	cosine	0.1	0.1	1024
MsViT-Meidum	ImageNet	300	8e-4	cosine	0.1	0.1	1024
MsViT-Base	ImageNet	150	8e-4	cosine	0.1	0.1	1024
MsViT-Meidum	ImageNet-21k	90	5e-4	cosine	0.1	0.1	1024
MsViT-Base	ImageNet-21k	90	5e-4	cosine	0.1	0.1	1024
MsViT-Meidum	ImageNet-384	10	[2, 4]*e-2	cosine	0.	0.1	512
MsViT-Base	ImageNet-384	10	[2, 4]*e-2	cosine	0.	0.1	512
Model	Dataset	iterations	Base Lr	LR decay	Weight decay	Drop Path	Batch size
MsViT-Tiny-1x	COCO	60k-80k-90k	1e-4	multi-step	0.05	[0.05, 0.1]	16
MsViT-Small-1x	COCO	60k-80k-90k	1e-4	multi-step	0.05	[0.1, 0.2]	16
MsViT-Meidum-D-1x	COCO	60k-80k-90k	1e-4	multi-step	0.05	[0.2, 0.3]	16
MsViT-Base-D-1x	COCO	60k-80k-90k	8e-5	multi-step	0.05	[0.2, 0.3]	16
MsViT-Tiny-3x+ms	COCO	180k-240k-270k	1e-4	multi-step	0.05	[0.05, 0.1]	16
MsViT-Small-3x+ms	COCO	180k-240k-270k	1e-4	multi-step	0.05	[0.1, 0.2]	16
MsViT-Meidum-D-3x+ms	COCO	180k-240k-270k	1e-4	multi-step	0.05	[0.2, 0.3]	16
MsViT-Base-D-3x+ms	COCO	180k-240k-270k	8e-5	multi-step	0.05	[0.2, 0.3]	16

Table 9. Hyperparameters for training. We use MsViT to represent the multi-scale vision transformers with different kinds of attention mechanisms, including our Vision Longformer (ViL). For the experiments trained on COCO, MsViT is combined with the Retinanet or Mask R-CNN. The training configs for Retinanet or Mask R-CNN are the same, and we still use MsViT for their unified short name. We do not apply gradient clipping for all ImageNet classification training and apply gradient clipping at global norm 1 for COCO object detection/segmentation. We use AdamW for all our experiments, except that we use SGD with momentum 0.9 for the ImageNet-384 fine-tuning experiments.

Model	Tiny		Small	
	CLS	Ave Pool	CLS	Ave Pool
DeiT/16[43]	72.2	-	79.8	-
+Layernorm	72.91	73.36	80.33	80.32
+2D Pos	73.21	73.09	80.44	80.75

Table 10. Ablation of the effects of LayerNorm and 2-D positional embedding in the patch embedding of the E-ViT module, with ImageNet Top-1 accuracy. The improvement over DeiT [43] comes from the added LayerNorm. The 2-D positional embedding is mainly for saving parameters for high-resolution feature maps. The column names of "CLS" and "Ave Pool" indicate how the image feature is obtained for the linear classification head.

Model	Tiny		Small	
	CLS	Ave Pool	CLS	Ave Pool
ViL-1,2,8,1-APE	75.72	75.98	81.65	81.99
ViL-1,1,8,2-APE	76.18	76.25	82.12	82.08

Table 11. For ViL models that has only one attention block in the last stage (ViL 1-2-8-1), the average pooled feature from all tokens works better than the feature of the CLS token. When there are more than 2 attention blocks in the last stage (ViL 1-1-8-2), the difference between these two features disappears.

pipelines. The results for the standard "3x+MS" schedule are reported in Table 6 in the main paper.

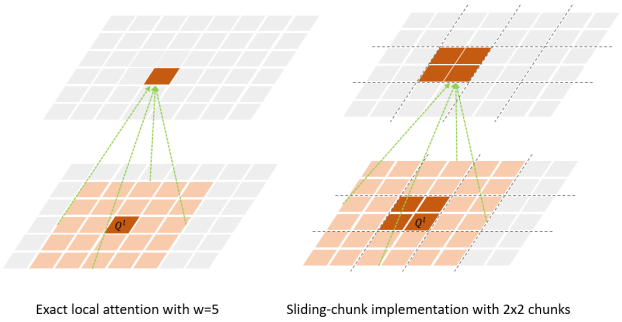


Figure 4. The sliding-chunk implementation of Vision Longformer. This implementation (Right) lets one token attends to more tokens than the exact conv-like local attention (Left). Our sliding-chunk implementation has the choice to be exactly the same with the conv-like local attention (Left), by masking out tokens that should not be attended to. For chunks on the boundaries, our implementation supports both no padding and cyclic padding.

C. Implementations and Efficiency of Vision Longformer In Practice

There is a trivial implementation of the conv-like sliding window attention, in which we compute the full quadratic attention and then mask out non-neighbor tokens. This approach suffers from the quadratic complexity w.r.t. number of tokens (quartic w.r.t. feature map size), and is impractical for real use, as shown by the blue curve in Figure 5. We

Flat Models	Tiny				Small			
DeiT / 16 [43]	72.2				79.8			
E-ViT(full $\times$ 12/16)	73.21 (CLS) / 73.09 (AVG)				80.44 (CLS) / 80.75 (AVG)			
Multi-scale	Tiny-3stage / 8		Tiny-4stage / 4		Small-3stage / 8		Small-4stage / 4	
Models	1-10-1	2-9-1	1-1-9-1	1-2-8-1	1-10-1	2-9-1	1-1-9-1	1-2-8-1
Full Attention	75.86	75.79	76.06	75.60	81.66	81.73	81.93*	81.91*
Vision Longformer	75.63$_{\pm 0.23}$	75.88$_{\pm 0.08}$	76.18$_{\pm 0.12}$	75.98$_{\pm 0.10}$	81.57	81.81	81.79	81.99
Linformer [46]	74.54	74.72	74.71	74.74	80.76	80.99	81.19	80.98
Partial Linformer	75.64	75.82	75.56	75.33	81.66	81.63	81.66	81.79
SRA/64 [47]	68.71	68.84	69.08	68.78	75.9	76.18	76.35	76.37
SRA/32 [47]	73.16	73.46	73.22	73.2	79.82	79.8	79.96	79.9
Partial SRA/32	75.17	75.8	75.2	75.26	81.63	81.59	81.62	81.61
Global	70.93	71.62	71.52	72.00	79.04	79.08	79.17	78.97
Partial Global	75.55	75.61	75.32	75.4	81.39	81.42	81.6	81.45
Performer	71.28	71.87	71.12	73.09	78.17	78.58	78.81	78.72
Partial Performer	75.65	75.74	75.34	75.93	81.59	81.86	81.72	81.72

Table 12. Overall comparison in ImageNet top-1 accuracy, with input size 224. Tiny-4stage / 4 means that the model has comparable size with DeiT-Tiny, has 4 stages and uses patch size 4x4 in the initial pixel space. 1-2-8-1 means that the model contains 4 stages, each stage has 1/2/8/1 MSA-FFN blocks, respectively. *Partial* means that the last two stages, which contain most of the attention blocks, still use full attention. Vision Longformer does not have *Partial* version because its window size is set as 15 (comparable with the ViT(DeiT)/16 feature map size 14), and its attention mechanism in the last two stages is equivalent to full attention. * indicates that the training batch size is 256 (with learning rate linearly scaled down), different from all other experiments with batch size 1024 in this table.

Flat Models	Tiny				Small			
DeiT / 16 [43]	5.7 (M) parameters, 1.3 GFLOPs				22.1 (M) parameters, 4.6 GFLOPs			
E-ViT(full $\times$ 12/16)	5.7 (M) parameters, 1.3 GFLOPs				22.1 (M) parameters, 4.6 GFLOPs			
Multi-scale	Tiny-3stage / 8		Tiny-4stage / 4		Small-3stage / 8		Small-4stage / 4	
Models	1-10-1	2-9-1	1-1-9-1	1-2-8-1	1-10-1	2-9-1	1-1-9-1	1-2-8-1
Full Attention	7.1, 1.35	6.8, 1.45	6.7, 2.29	6.4, 2.39	27.6, 4.84	26.3, 5.05	26.0, 6.74	24.6, 6.95
Vision Longformer	7.1, 1.27	6.8, 1.29	6.7, 1.33	6.4, 1.35	27.6, 4.67	26.3, 4.71	26.0, 4.82	24.6, 4.86
Linformer [46]	7.8, 1.57	7.7, 1.6	8.2, 1.69	8.0, 1.73	28.3, 5.27	27.1, 5.35	27.4, 5.55	26.3, 5.62
Partial Linformer	7.3, 1.31	7.2, 1.37	7.7, 1.46	7.6, 1.52	27.8, 4.76	26.7, 4.88	27.0, 5.08	25.8, 5.21
SRA/64 [47]	14.2, 0.99	13.9, 0.99	13.8, 1.0	13.5, 1.0	55.9, 3.92	54.6, 3.92	54.3, 3.97	52.9, 3.97
SRA/32 [47]	8.7, 1.09	8.4, 1.09	8.3, 1.1	8.0, 1.1	34.1, 4.23	32.7, 4.23	32.5, 4.28	31.1, 4.28
Partial SRA/32	7.3, 1.23	7.1, 1.22	7.0, 1.24	6.8, 1.22	28.2, 4.6	27.4, 4.56	27.1, 4.61	26.4, 4.57
Global	7.2, 1.69	6.9, 1.7	6.8, 1.75	6.5, 1.76	27.8, 6.65	26.4, 6.67	26.2, 6.76	24.9, 6.78
Partial Global	7.2, 1.6	6.9, 1.62	6.8, 1.68	6.5, 1.7	27.8, 6.07	26.4, 6.14	26.2, 6.23	24.9, 6.3
Performer	7.3, 1.81	7.0, 1.86	6.9, 1.99	6.6, 2.05	27.8, 5.75	26.5, 5.87	26.2, 6.14	24.8, 6.26
Partial Performer	7.1, 1.35	6.8, 1.45	6.7, 1.57	6.4, 1.67	27.6, 4.84	26.3, 5.04	26.0, 5.31	24.7, 5.52

Table 13. Overall comparison in number of parameters (M) and GFLOPs, with input size 224. Tiny-4stage / 4 means that the model has comparable size with DeiT-Tiny, has 4 stages and uses patch size 4x4 in the initial pixel space. 1-2-8-1 means that the model contains 4 stages, each stage has 1/2/8/1 MSA-FFN blocks, respectively. *Partial* means that the last two stages, which contain most of the attention blocks, still use full attention. Vision Longformer does not have *Partial* version because its window size is set as 15 (comparable with the ViT(DeiT)/16 feature map size 14), and its attention mechanism in the last two stages is equivalent to full attention.

Backbone	#Params (M)	FLOPs (G)	RetinaNet 1x schedule						Mask R-CNN 1x schedule					
			AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}
ResNet18	21.3/31.2	190/207	31.8	49.6	33.6	16.3	34.3	43.2	34.0	54.0	36.7	31.2	51.0	32.7
PVT-Tiny[47]	23.0/32.9	–/–	36.7	56.9	38.9	22.6	38.8	50.0	36.7	59.2	39.3	35.1	56.7	37.3
ViL-Tiny-RPB	16.6/26.9	183/199	40.8	61.3	43.6	26.7	44.9	53.6	41.4	63.5	45.0	38.1	60.3	40.8
ResNet50	37.7/44.2	239/260	36.3	55.3	38.6	19.3	40.0	48.8	38.0	58.6	41.4	34.4	55.1	36.7
PVT-Small[47]	34.2/44.1	226/245	40.4	61.3	43.0	25.0	42.9	55.7	40.4	62.9	43.8	37.8	60.1	40.3
ViL-Small-RPB	35.7/45.0	255/277	44.2	65.2	47.6	28.8	48.0	57.8	44.9	67.1	49.3	41.0	64.2	44.1
ResNet101	56.7/63.2	315/336	38.5	57.8	41.2	21.4	42.6	51.1	40.4	61.1	44.2	36.4	57.7	38.8
ResNeXt101-32x4d	56.4/62.8	319/340	39.9	59.6	42.7	22.3	44.2	52.5	41.9	62.5	45.9	37.5	59.4	40.2
PVT-Medium[47]	53.9/63.9	283/302	41.9	63.1	44.3	25.0	44.9	57.6	42.0	64.4	45.6	39.0	61.6	42.1
ViL-Medium-RPB	50.8/60.1	330/352	46.8	68.1	50.0	31.4	50.8	60.8	47.6	69.8	52.1	43.0	66.9	46.6
ResNeXt101-64x4d	95.5/101.9	473/493	41.0	60.9	44.0	23.9	45.2	54.0	42.8	63.8	47.3	38.4	60.6	41.3
PVT-Large[47]	71.1/81.0	345/364	42.6	63.7	45.4	25.8	46.0	58.4	42.9	65.0	46.6	39.5	61.9	42.5
ViL-Base-RPB	66.7/76.1	421/439	47.8	69.2	51.4	32.4	52.3	61.8	48.6	70.5	53.4	43.6	67.6	47.1

Table 14. Object detection and instance segmentation performance on the COCO val2017. The numbers before and after "/" at column 2 and 3 are the model size and complexity for RetinaNet and Mask R-CNN, respectively. The FLOPs (G) are measured at resolution 800×1333. "–" means data publicly unavailable. Our ViL-Tiny and ViL-Small models are pre-trained on ImageNet-1K, our ViL-Medium and ViL-Base models are pre-trained on ImageNet-21k. ViL results are highlighted with gray background.

only use it to verify the correctness of our other implementations.

We have implemented Vision Longformer in three ways:

1. Using Pytorch's unfold function. We have two subversions: one using nn.functional.unfold (denoted as "unfold/nn.F") and the other using tensor.unfold (denoted as "unfold/tensor"). As shown in Figure 5, the "unfold/tensor" version (red solid line) is more efficient both in time and memory than the "unfold/nn.F" version (red dotted line). However, both of them are even slower and use more memory than the full attention!

2. Using a customized CUDA kernel, denoted as "cuda_kernel". We make use of the TVM, like what has done in Longformer [3], to write a customized CUDA kernel for Vision Longformer. As shown in Figure 5, the "cuda_kernel" (green line) achieves the theoretical optimal memory usage. Its time complexity is also reduced to linear w.r.t. number of tokens (quadratic w.r.t. feature map size). However, since it's not making use of the highly optimized matrix multiplication libraries in CUDA, it's speed is still slow in practice.

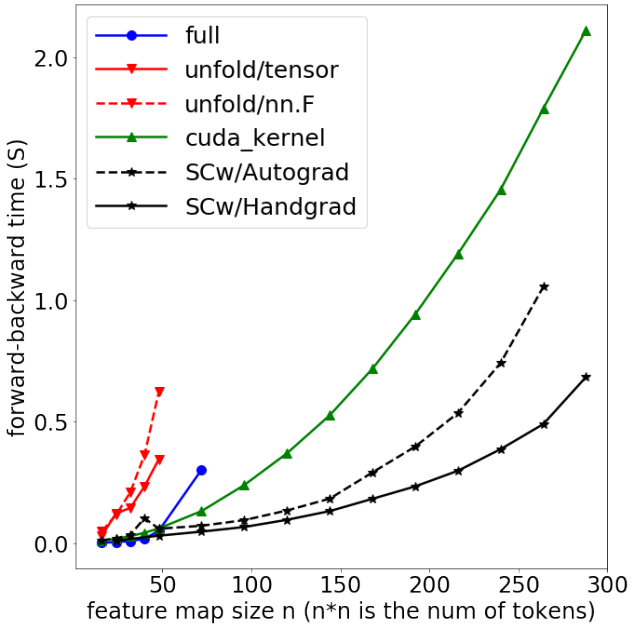
3. Using a sliding chunk approach, illustrated in Figure 4. For this sliding chunk approach, we have two subversions: one using Pytorch's autograd to compute backward step (denoted as "SCw/Autograd") and the other writing a customized torch.autograd.Function with hand-written backward function (denoted as "SCw/Handgrad"). Both sub versions of this sliding chunk approach are fully implemented with Pytorch functions and thus make use of highly optimized matrix multiplication libraries in CUDA. As shown in Figure 5, both of them are faster than the "cuda_kernel" implementation.

In the sliding chunk approach, to achieve a conv-like local attention mechanism with window size $2w + 1$, we split the feature map into chunks with size $w \times w$. Each chunk only attends to itself and its 8 neighbor chunks. The Pytorch Autograd will save 9 copies of the feature map (9 nodes in the computing graph) for automatic back-propagation, which is not time/memory efficient. The "SCw/Handgrad" version defines a customized torch.autograd.Function with hand-written backward function, which greatly saves the memory usage and also speeds up the algorithm, as shown in Figure 5. We would like to point out that the memory usage of the "SCw/Handgrad" version is nearly optimal (very close to that of the "cuda_kernel"). Similar speed-memory trade-off with different implementations of local attention mechanism has been observed in the 1-D Longformer [3], too; see Figure 1 in [3]. We would like to point out that Image Transformer [30] has an implementation of of 2-D conv-like local attention mechanism, which is similar to our "SCw/Autograd" version. The Image Transformer [30] applies it to the image generation task.

This sliding-chunk implementation (Figure 4 Right) lets one token attends to more tokens than the exact conv-like local attention (Figure 4 Left). Our sliding-chunk implementation has the choice to be

1. exactly the same with the conv-like local attention (Left), by masking out tokens that should not be attended to,

2. sliding chunk without padding, in which the chunks on the boundary have less chunks to attend to,



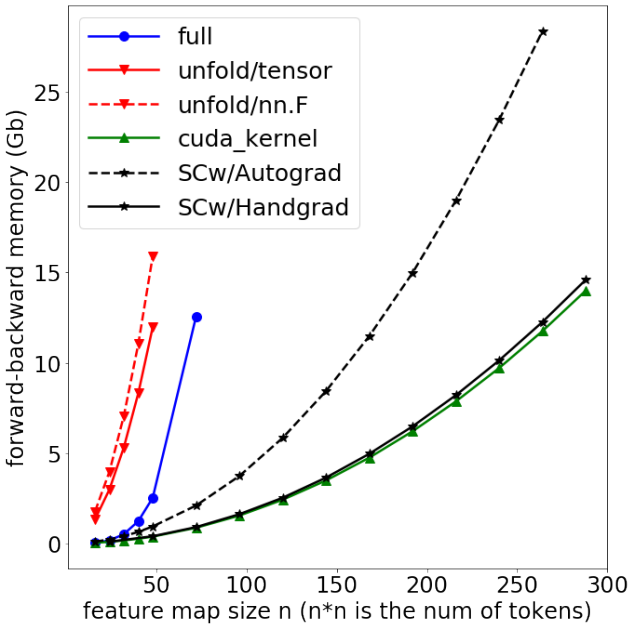
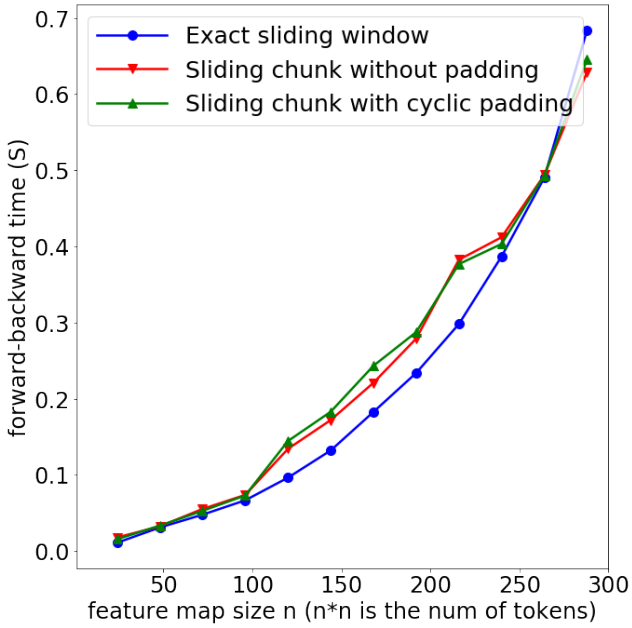


Figure 5. Compare of running time (including forward and backward) and memory usage of different implementations of the conv-like attention in Vision Longformer. All of these implementations shown in the figures are mathematically equivalent, doing the exact conv-like sliding window attention with window size 17.



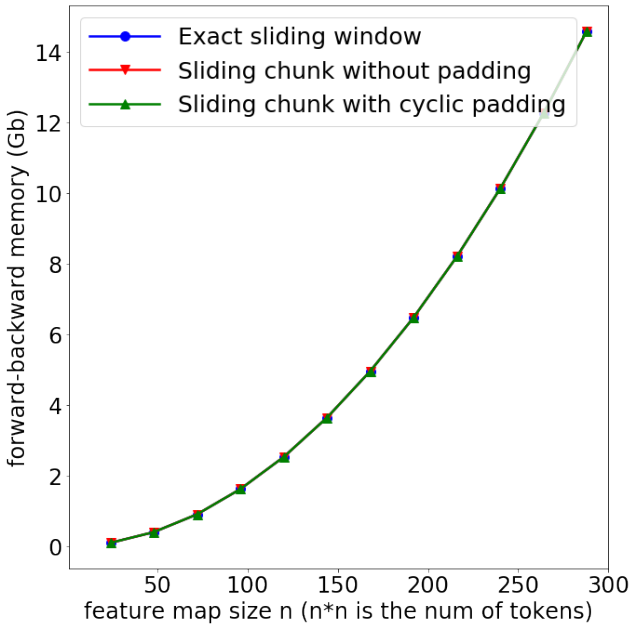


Figure 6. Compare of three masking methods of our "SCw/Handgrad" implementation of conv-like local attention: exact conv-like sliding window attention, sliding chunk attention without padding for boundary chunks, and sliding chunk attention with cyclic padding for boundary chunks. The are nearly the same in terms of running time (including forward and backward) and memory usage. The window size is 17 and thus chunk size is 8.

3. sliding chunk with cyclic padding, in which the chunks on the boundary still attend to 9 chunks with cyclic padded chunks.

Since these three masking methods only differ by the attention masks to mask out invalid tokens, their speed and memory usage are nearly the same, as shown in Figure 6. For ImageNet classification, we observe no obvious difference in top1 accuracy between "exact sliding window" and "sliding chunk without padding", while "sliding chunk with cyclic padding" performs slightly worse most of the time. For object detection, we observe that "sliding chunk without padding" performs consistently better than "exact sliding

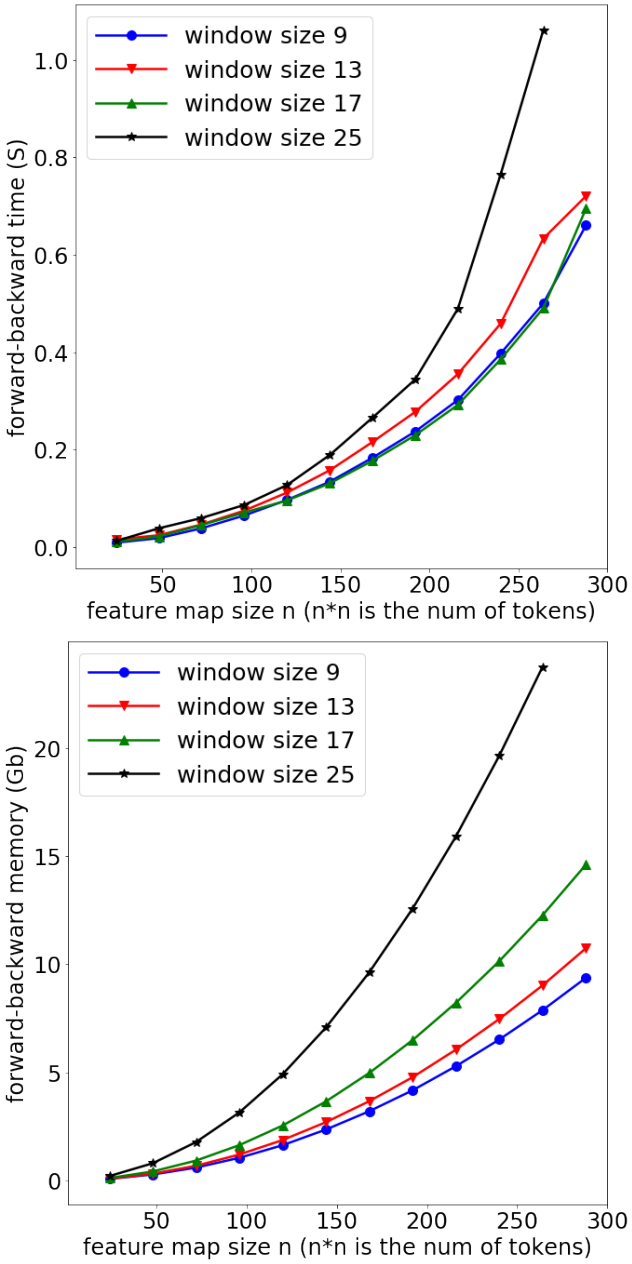


Figure 7. Running time (including forward and backward) and memory usage of our "SCw/Handgrad" implementation of conv-like local attention (sliding chunk attention without padding mode) with different window sizes. The speed is not sensitive to the window size for small window sizes ($\leq$ 17) and the memory usage monotonically increases.

window", as shown in Figure 8. Therefore, we make "sliding chunk without padding" as the default making method for Vision Longformer, although it sacrifices some translational invariance compared with "exact sliding window".

In Figure 7, we show the running time (including forward and backward) and memory usage of our

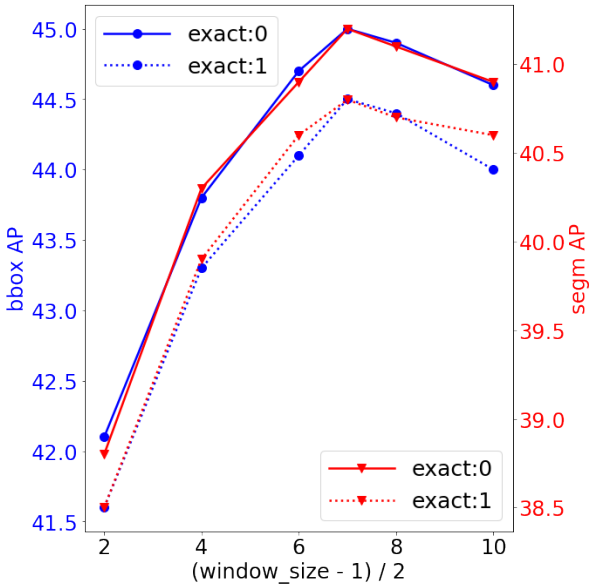


Figure 8. "Sliding chunk without padding" performs consistently better than "exact sliding window" for object detection with Mask R-CNN. All use the same ImageNet1K pre-trained checkpoint (ViL-Small-RPB in Table 4).

"SCw/Handgrad" implementation of conv-like local attention (sliding chunk attention without padding mode) with different window sizes. We can see that the speed is not sensitive to the window size for small window sizes ($\leq$ 17) and the memory usage monotonically increases.

Finally, both the "unfold/nn.F" and the "cuda_kernel" implementations support dilated conv-like attention. The customized CUDA kernel is even more flexible to support different dilations for different heads. The sliding-chunk implementation does not support this dilated conv-like attention. In this paper, we always use the sliding-chunk implementation due to its superior speed and nearly optimal memory complexity.

In Figure 5, 6 and 7, the evaluation is performed on a single multi-head self-attention module (MSA) with the conv-like local attention mechanism, instead of on the full multi-scale Vision Longformer. With this evaluation, we can clearly see the difference among different implementations of the conv-like local attention mechanism.

D. Random-shifting strategy to improve training efficiency

We propose the random-shifting training strategy for Vision Longformer, to further accelerate the training speed of Vision Longformer. More specifically, instead of attending to all 8 neighbor patches, one patch can only attend to itself and one random neighbor patch during training. To achieve this, we define 10 modes of the sliding-chunk local attention:

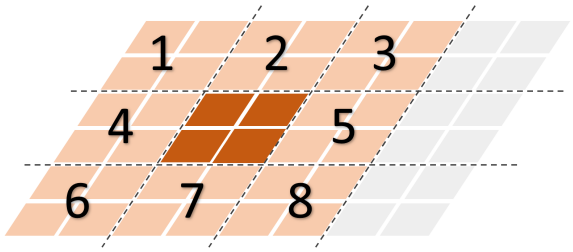


Figure 9. Illustration of the 8 modes in the random-shifting training strategy. For mode i ($1 <= i <= 8$), the query chunk (dark brown) attends to itself and the i'th neighbor chunk.

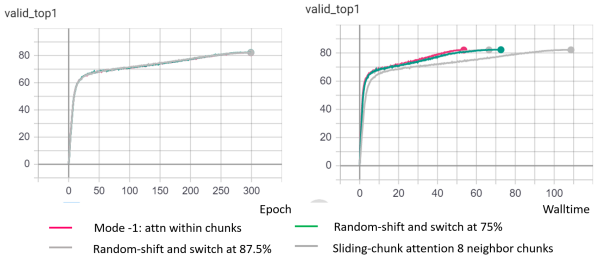


Figure 10. The random-shifting strategy does not harm the model performance (Left), an accelerates the Vision Longformer training significantly (Right). When zooming in, the performance of "random-shift and switch at 75%" is slightly better than the "Sliding-chunk attention with 8 neighbor chunks".

- 0 (default): attend to itself and all 8 neighbor chunks,
- -1 : only attend to itself chunk,
- i ($1 <= i <= 8$) : attend to itself chunk and the i'th neighbor chunk.

The ordering of the 8 neighbor patches is visualized in Figure 9. During training, we can randomly sample one mode from 1 to 8 and perform the corresponding random-shifting attention. We switch from the random-shifting mode to the default 8-neighbor mode after $x\%$ training iterations, and this switch time $x\%$ is a hyper-parameter with default value 75%. This switch, can be seen as fine-tuning, is necessary to mitigate the difference of model's behavior during training and inference. As shown in Figure 10, this random-shifting training strategy accelerates the Vision Longformer training significantly, while not harming the final model performance.

E. Other Efficient Attention Mechanisms utilized in this work

In this paper, we compare Vision Longformer with the following alternative choices of efficient attention methods.

Pure global memory (a = global). In Vision Longformer, see Figure 2 (Left), if we remove the local-to-local attention, then we obtain the pure global memory attention mechanism (called Global Attention hereafter). Its memory complexity is $\mathcal{O}(n_g(n_g + n_l))$, which is also linear w.r.t. n_l. However, for this pure global memory attention, n_g has to be much larger than 1. In practice, we set different numbers of global tokens for different stages, as shown in Table 15, with more global tokens in the first 2 stages and less in the last 2 stages. This setting makes the memory/computation complexity comparable with other attention mechanisms under the same model size.

Linformer[46] (a = LIN) projects the $n_l \times d$ dimensional keys and values to $K \times d$ dimensions using additional projection layers, where $K \ll n_l$. Then the n_l queries only attend to these projected K key-value pairs. The memory complexity of Linformer is $\mathcal{O}(Kn_l)$. We gradually increase K (by 2 each time) and its performance gets nearly saturated at 256. Therefore, $K = 256$ is our choice for this Linformer attention, which turns out to be the same with the recommended value. Notice that Linformer's projection layer (of dimension $K \times n_l$) is specific to the current n_l, and cannot be transferred to higher-resolution tasks that have a different n_l. It is possible to transfer Linformer's weight by resizing feature maps of a different size to the original feature map size that Linformer is trained with and then applying the Linformer's projection. We do not explore this choice in this work.

Spatial Reduction Attention (SRA) [47] (a = SRA) is similar to Linformer, but uses a convolution layer with kernel size R and stride R to project the key-value pairs, hence resulting in n_l/R^2 compressed key-value pairs. Therefore, The memory complexity of SRA is $\mathcal{O}(n_l^2/R^2)$, which is still quadratic w.r.t. n_l but with a much smaller constant $1/R^2$. When transferring the ImageNet-pretrained SRA-models to high-resolution tasks, SRA still suffers from the quartic computation/memory blow-up w.r.t. the feature map resolution. Pyramid Vision Transformer [47] uses this SRA to build multi-scale vision transformer backbones, with different spatial reduction ratios ($R_1 = 8, R_2 = 4, R_3 = 2, R_4 = 1$) for each stage. With this PVT's setting, the key and value feature maps at all stages are essentially with resolution $\frac{H}{32} \times \frac{W}{32}$. This choice is able to scale up to image resolution 600×1000, but the memory usage is much larger than ResNet counterparts for 800×1333.

In this paper, we benchmarked the performance of SRA/32 with SR ratios $R_1 = 8, R_2 = 4, R_3 = 2, R_4 = 1$ (same as PVT [47]) and SRA/64 with SR ratios $R_1 = 16, R_2 = 8, R_3 = 4, R_4 = 2$ (two times more downsizing from that in PVT [47]), as shown in Table 15. The SRA/64 setting makes the memory usage comparable with other efficient attention mechanisms under the same model size, but introduces more parameters due to doubling the kernel size

of the convolutional projection layer.

Performer [10] (a = performer) uses random kernel approximations to approximate the Softmax computation in MSA, and achieves a linear computation/memory complexity with respect to n_l and the number of random features K. We use the default $K = 256$ orthogonal random features (OR) for Performer. The memory/space complexity of performer is $\mathcal{O}(Kd + n_l d + Kn_l)$ while its computation/time complexity is $\mathcal{O}(Kn_l d)$. For the time complexity, we ignore the complexity of generating the orthogonal random features, which in practice cannot be ignored during training. We refer to Section B.3 in [10] for a detailed discussion of theoretical computation/memory complexity of Performer.

One important technique in training Performer is to redraw the random features during training. In our ImageNet classification training, we adopt a heuristic adaptive redrawing schedule: redraw every $1 + 5T$ iterations in Epoch T ($T = 0, 1, ..., 299$). In our COCO object detection/segmentation training, the Performer is initialized from ImageNet pretrained checkpoint and thus there is no need to redraw very frequently in the initial training stage.Therefore, we redraw the random features every 1000 iterations in COCO object detection/segmentation training.

Model	Stage1	Stage2	Stage3	Stage4
	Window size w			
ViL-3stage	15		15	15
ViL-4stage	15	15	15	15
ViL-4stage (384)	13	17	25	25
	Number of global tokens n_g			
Global-3stage	256		64	16
Global-4stage	256	256	64	16
	Projection Dimension K			
Linformer-3stage	256		256	256
Linformer-4stage	256	256	256	256
	Spatial reduction ratio R			
SRA/64-3stage	8		4	2
SRA/64-4stage	16	8	4	2
SRA/32-3stage	4		2	1
SRA/32-4stage	8	4	2	1
	Number of orthogonal random features K			
Performer-3stage	256		256	256
Performer-4stage	256	256	256	256

Table 15. Attention mechanism specific hyper-parameters. See Appendix E for the details of these attention mechanisms.

F. Visualization of DeiT's attention maps to validate the inductive bias in Vision Longformer

We visualize the attention maps from DeiT's pretrained models to argue that the "local attention + global memory" mechanism in Vision Longformer is a reasonable inductive bias for vision transformers. More specifically, we take the pretrained DeiT-Tiny model from https://dl.fbaipublicfiles.com/deit/deit_tiny_patch16_224-a1311bcf.pth, infer it on images from ImageNet validation set, and display its attention maps of the first attention block in Figure 11, 12 and 13, for head 0, head 1 and head 2, respectively.

In Figure 11, the attention map of head 0 is just like the 2-D convolution, focusing on neighboring patches. There are also relative large weights on the CLS token. In Figure 13, the attention map of head 0 is just like the "global memory" mechanism, with every local token attends to the global CLS token. Each local token also attends to itself, like "local attention" with window size 1. In Figure 12, the attention map of head 1 seems to be global, with large attention scores on visually similar patches (i.e., the dog's white fur, the gray wall).

These three attention patterns for these three heads are consistent for different images. For other DeiT pretrained models (like DeiT-Small and DeiT-Base), their attention maps in the first attention block can also be classified into these three attention patterns. Longformer exactly captures the "local attention" and the "global memory" patterns, and may introduce some information loss for the "global-similarity" pattern. In comparison with the full attention, the Vision Longformer attention does not lead to any accuracy drop in low-resolution ImageNet classification problem (Table 2) and leads to small accuracy drop in high-resolution Object Detection/Segmentation tasks (Table 7).

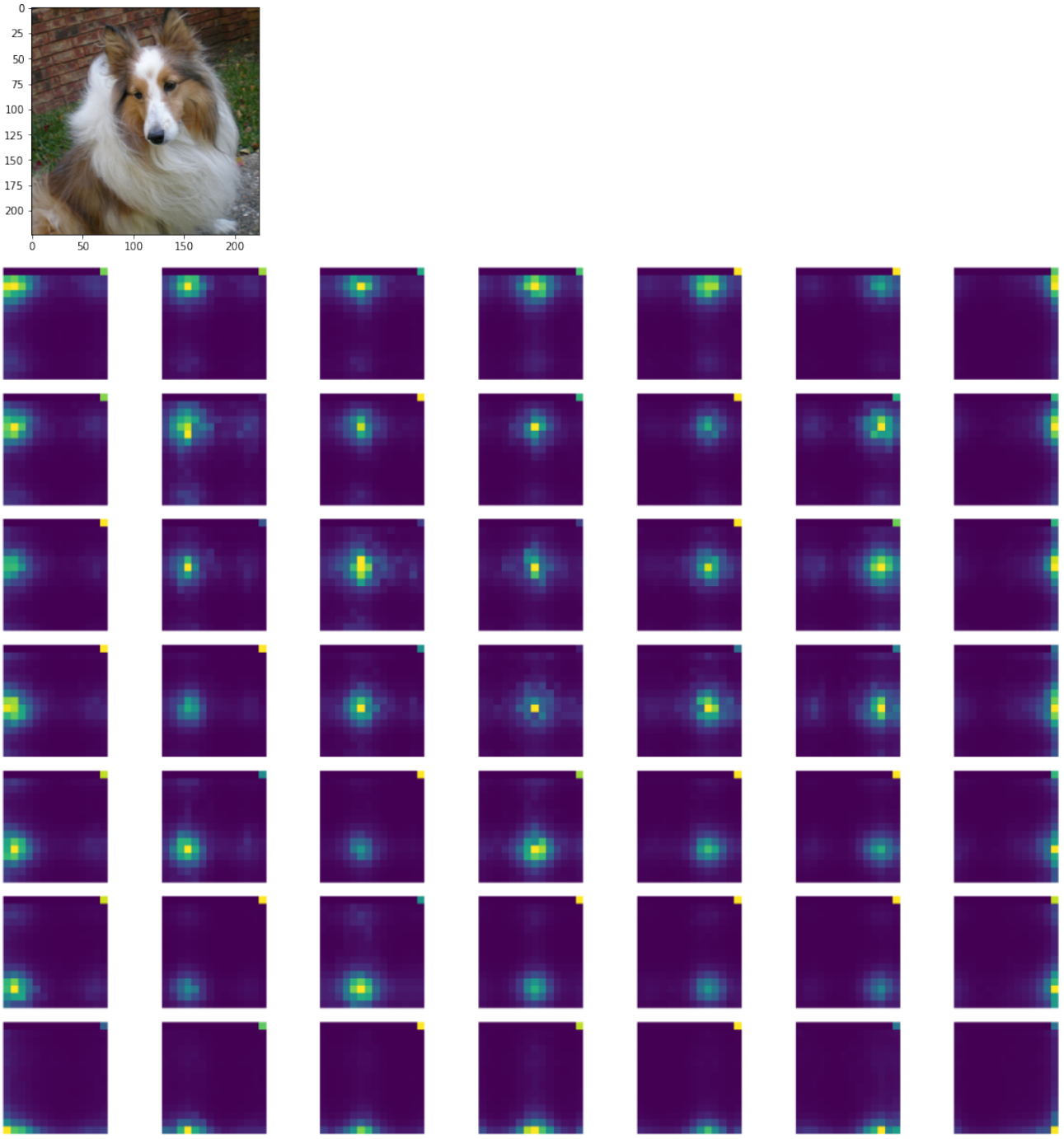


Figure 11. Attention map of DeiT-Tiny at Block 0, head 0. This head's attention map has exactly the "local attention + global memory" pattern in Vision Longformer, with more emphasis on the "local attention". The (i, j) subplot shows the attention map of the local token $(2 * i, 2 * j)$ as query. Each attention map A is a matrix of size 15×14, where $A[1:15, 0:14]$ is the attention scores to the 14×14 local tokens and $A[0, 13]$ (top-right corner) shows the attention score to the CLS token. $A[0, 0:13]$ are zeros.

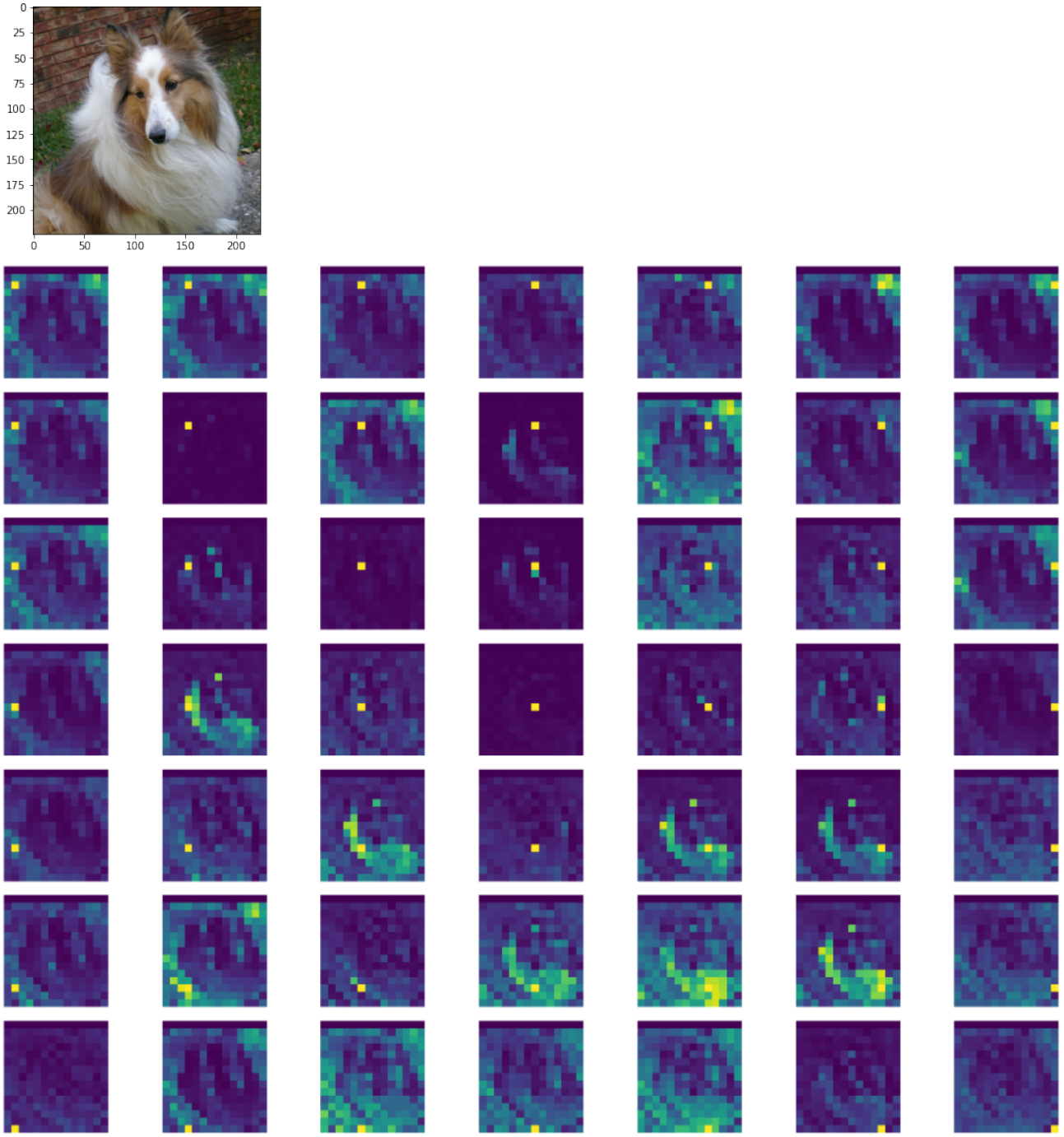


Figure 12. DeiT-Tiny: Block 0, head 1. This head's attention map seems to be global, with large attention scores on visually similar patches (i.e., the dog's white fur, the gray wall). The (i, j) subplot shows the attention map of the local token $(2 * i, 2 * j)$ as query. Each attention map A is a matrix of size 15×14, where $A[1:15, 0:14]$ is the attention scores to the 14×14 local tokens and $A[0, 13]$ (top-right corner) shows the attention score to the CLS token. $A[0, 0:13]$ are zeros.

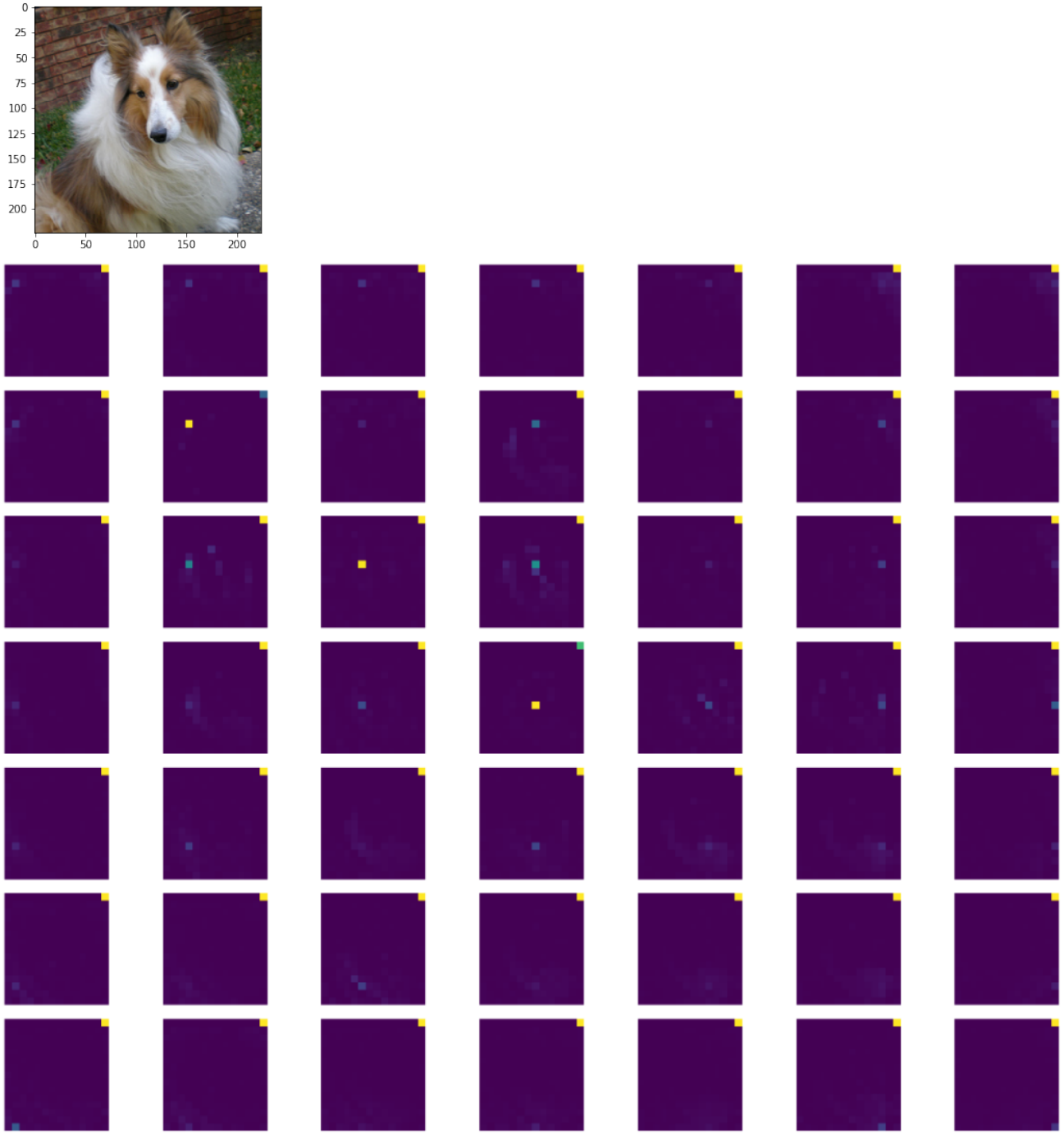


Figure 13. DeiT-Tiny: Block 0, head 2. This head's attention map has exactly the "local attention + global memory" pattern in Vision Longformer, with more emphasis on the "global memory". The (i, j) subplot shows the attention map of the local token $(2 * i, 2 * j)$ as query. Each attention map A is a matrix of size 15×14, where $A[1 : 15, 0 : 14]$ is the attention scores to the 14×14 local tokens and $A[0, 13]$ (top-right corner) shows the attention score to the CLS token. $A[0, 0 : 13]$ are zeros.