Limitations of this work are larger scale models, more vision tasks, further optimization of accuracy, power and parameter amount, optimization of training consumption caused by multiple timesteps, etc., and we will work on them in future work. The experimental results in this paper are reproducible. We explain the details of model training and configuration in the main text and supplement it in the appendix. Our codes and models of Meta-SpikeFormer will be available on GitHub after review. Moreover, in this work, the designed meta SNN architecture is tested on vision tasks. For language tasks, the challenges faced will be different, such as parallel spiking neuron design, long-term dependency modeling in the temporal dimension, pre-training, architecture design, etc. need to be considered. This work can at least provide positive inspiration for SNN processing language tasks in long-term dependency modeling and architecture design, and we are working in this direction.

A Spike-Driven Self-Attention (SDSA) Operators

In this Section, we understand vanilla and spike-driven self-attention from the perspective of computational complexity.

A.1 Vanilla Self-Attention (VSA)

Given a float-point input sequence $X \in \mathbb{R}^{N \times D}$, float-point Query ($Q$), Key ($K$), and Value ($V$) in $\mathbb{R}^{N \times D}$ are calculated by three learnable linear matrices, where N is the token number, D is the channel dimension. The vanilla scaled dot-product self-attention is computed as (Dosovitskiy et al., 2021):

$$\mathrm{VSA}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V, \tag{16}$$

where $d = D/H$ is the feature dimension of one head and H is the head number, $\sqrt{d}$ is the scale factor. Generally, VSA performs multi-head self-attention, i.e., divide Q, K, V into H heads in the channel dimension. In the i-th head, Q^i, K^i, V^i in $\mathbb{R}^{N \times D/H}$. After the self-attention operation is performed on the H heads respectively, the outputs are concatenated together.

In VSA, Q and K are matrix multiplied first, and then their output is matrix multiplied with V. The computational complexity of $\mathrm{VSA}(\cdot)$ is $O(N^2 D)$, which has a *quadratic* relationship with the toke number N.

A.2 Spike-Driven Self-Attention (SDSA)

In our Transformer-based SNN blocks, as shown in Fig. 2, given a spike input sequence $S \in \mathbb{R}^{T \times N \times D}$, spike-form (binary) Q_S, K_S, and V_S in $\mathbb{R}^{T \times N \times D}$ are calculated by three learnable re-parameterization convolutions Ding et al. (2021) with 3×3 kernel size:

$$Q_S = \mathcal{SN}(\mathrm{RepConv}_1(U)), K_S = \mathcal{SN}(\mathrm{RepConv}_2(U)), V_S = \mathcal{SN}(\mathrm{RepConv}_3(U)), \tag{17}$$

where $\mathrm{RepConv}(\cdot)$ denotes the re-parameterization convolution, $\mathcal{SN}(\cdot)$ is the spiking neuron layer. For the convenience of mathematical expression, we assume $T = 1$ in the subsequent formulas.

SDSA-1. The leftmost SDSA-1 in Fig. 3 is the operator proposed in Spike-driven Transformer (Yao et al., 2023b). The highlight of SDSA-1 is that the matrix multiplication between Q_S, K_S, V_S in SDSA is replaced by Hadamard product:

$$\mathrm{SDSA}_1(Q_S, K_S, V_S) = Q_S \otimes \mathcal{SN}\left(\mathrm{SUM_c}\left(K_S \otimes V_S\right)\right), \tag{18}$$

where $\otimes$ is the Hadamard product, $\mathrm{SUM_c}(\cdot)$ represents the sum of each column, and its output is a D-dimensional row vector. The Hadamard product between spike tensors is equivalent to the mask operation. Compared to the VSA in Eq. 16, $\mathrm{SUM_c}(\cdot)$ and $\mathcal{SN}(\cdot)$ take the role of softmax and scale.

Now, we analyze the computational complexity of SDSA-1. Before that, we would like to introduce the concept of *linear attention*. If the softmax in VSA is removed, K and V can be multiplied first, and then their output is matrix multiplied with Q. The computational complexity becomes $O(ND^2/H)$, which has a linear relationship with the toke number N. This variant of attention is called linear attention (Katharopoulos et al., 2020). Further, consider an extreme case in linear attention, set

$H = D$. That is, in each head, Q^i, K^i, V^i in $\mathbb{R}^{N\times 1}$. Then, the computational complexity is $O(ND)$, which has a linear relationship with both the toke number N and the channel dimension D. This variant of linear attention is called *hydra attention* (Bolya et al., 2023).

SDSA-1 has the same computational complexity as hydra attention, i.e., $O(ND)$. Firstly, K_S and V_S in Eq. 18 participate in the operation first, thus it is a kind of linear attention. Further, we consider the special operation of Hadamard product. Assume that the i-th column vectors in K_S and V_S are a and b respectively. Taking the Hadamard product of a and b and summing them is equivalent to multiplying b times the transpose of a, i.e., $\mathrm{SUM_c}(a \otimes b) = a^T b$. In total, there are D times of dot multiplication between vectors, and N additions are performed each time. Thus, the computational complexity of SDSA-1 is $O(ND)$, which is consistent with hydra attention (Bolya et al., 2023).

SDSA-2. SDSA-1 in Eq. 18 actually first uses Q_S and K_S to calculate the binary self-attention scores, and then performs feature masking on V_S in the channel dimension. We can also get the binary attention scores using only Q_S, i.e., SDSA-2 is presented as:

$$\mathrm{SDSA}_2(Q_S, V_S) = \mathcal{SN}\left(\mathrm{SUM_c}\left(Q_S\right)\right) \otimes V_S. \tag{19}$$

We evaluate SDSA-1 and SDSA-2 in Table 5. Specifically, SDSA-1-based Meta-SpikeFormer vs. SDSA-2-based Meta-SpikeFormer: Param, 31.3M vs. 28.6M; Power, 7.2mJ vs. 6.3mJ; Acc, 74.6% vs. 74.2%. It can be seen that with the support of the Meta-SpikeFormer architecture, even if the Key matrix K_S is removed, the accuracy is only lost by 0.4%. The number of parameters and energy consumption are reduced by 8.7% and 12.5% respectively. Since the Hadamard product between spiking tensors Q_S and K_S in SDSA-1 can be regarded as a mask operation without energy cost, SDSA-1 and SDSA-2 have the same computational complexity, i.e., $O(ND)$. SDSA-2-based Meta-SpikeFormer has fewer parameters and power because there is no need to generate K_S.

SDSA-3 is the spike-driven self-attention operator used by default in this work, which is presented as:

$$\mathrm{SDSA}_3(Q_S, K_S, V_S) = \mathcal{SN}_s\left(Q_S\left(K_S^{\mathrm{T}} V_S\right)\right) = \mathcal{SN}_s((Q_S K_S^{\mathrm{T}})V_S). \tag{20}$$

In theory, the time complexity of $Q_S(K_S^{\mathrm{T}}V_S)$ and $(Q_S K_S^{\mathrm{T}})V_S$ are $O(N^2D)$ and $O(ND^2)$, respectively. The latter has a linear relationship with N, thus SDSA-3 is also a linear attention. Since $Q_S K_S^{\mathrm{T}} V_S$ yields large integers, a scale multiplication s for normalization is needed to avoid gradient vanishing. In our SDSA-3, we incorporate the s into the threshold of the spiking neuron to circumvent the multiplication by s. That is, the threshold in Eq. 20 is $s \cdot u_{th}$. We write such a spiking neuron layer with threshold $s \cdot u_{th}$ as $\mathcal{SN}_s(\cdot)$.

SDSA-4. On the basis of SDSA-3, we directly set the threshold of $\mathcal{SN}(\cdot)$ in Eq. 15 as a learnable parameter, and its initialization value is $s \cdot u_{th}$. We have experimentally found that the performance of SDSA-3 and SDSA-4 is almost the same (see Table 5). SDSA-4 consumes 0.1mJ less energy than SDSA-3 because the network spiking firing rate in SDSA-4 is slightly smaller than that in SDSA-3.

A.3 Discussion about SDSA operators

Compared with vanilla self-attention, the Q_S, K_S, V_S matrices of spike-driven self-attention are in the form of binary spikes, and the operations between Q_S, K_S, V_S do not include softmax and scale. Since there is no softmax and K_S and V_S can be computed first, spike-driven self-attention must be linear attention. This is the natural advantage of a spiking Transformer. On the other hand, in the current SDSA design, the operation between Q_S, K_S, V_S is Hadamard product or matrix multiplication, both of which can be converted into sparse addition operations. Therefore, SDSA not only has low computational complexity, but also only has sparse addition. Its energy consumption is much lower than that of vanilla self-attention (see Appendix B).

In Yu et al. (2022a;b), the authors summarized various ViT variants and argued that there is general architecture abstracted from ViTs by not specifying the token mixer (self-attention). This paper experimentally verifies that this view also holds true in Transformer-based SNNs. In Table 5, we tested four SDSA operators and found that the performance changes between SDSA-1/2/3/4 were not large (less than 1.2%). We expect the SNN domain to design more powerful SDSA operators in the future, e.g., borrowing from Swin (Liu et al., 2021), hierarchical attention (Hatamizadeh et al., 2023), and so on.

Table 6: FLOPs of self-attention modules. The FLOPs in VSA and SDSA are multiplied by $E_{MAC} = 4.6pJ$ and $E_{AC} = 0.9pJ$ respectively to obtain the final energy cost. R_C, $\widehat{R}$ denote the sum of spike firing rates of various spike matrices.

	VSA	SDSA-1	SDSA-2	SDSA-3	SDSA-4
Q, K, V	$3ND^2$	$T \cdot R_C \cdot 3 \cdot FL_{Conv}$	$T \cdot R_C \cdot 2 \cdot FL_{Conv}$	$T \cdot R_C \cdot 3 \cdot FL_{Conv}$	$T \cdot R_C \cdot 3 \cdot FL_{Conv}$
$f(Q, K, V)$	$2N^2D$	$T \cdot \widehat{R} \cdot ND$	$T \cdot \widehat{R} \cdot ND$	$T \cdot \widehat{R} \cdot ND^2$	$T \cdot \widehat{R} \cdot ND^2$
Scale	N^2	-	-	-	-
Softmax	$2N^2$	-	-	-	-
Linear	FL_{MLP}	$T \cdot R_C \cdot FL_{Conv}$	$T \cdot R_C \cdot FL_{Conv}$	$T \cdot R_C \cdot FL_{Conv}$	$T \cdot R_C \cdot FL_{Conv}$

B Theoretical Energy Evaluation

B.1 Spike-driven operators in SNNs

Spike-driven operators for SNNs are fundamental to low-power neuromorphic computing. In CNN-based SNNs, spike-driven Conv and MLP constitute the entire network. Specifically, the matrix multiplication between the weight and spike matrix in spike-driven Conv and MLP is transformed into sparse addition, which is implemented as addressable addition in neuromorphic chips (Frenkel et al., 2023).

By contrast, Q_S, K_S, V_S in spike-driven self-attention involve two matrix multiplications. One way is to execute element-wise multiplication between Q_S, K_S, V_S, like SDSA-1 in (Yao et al., 2023b) and SDSA-2 in this work (Eq. 19). And, element multiplication in SNNs is equivalent to mask operation with no energy cost. Another method is to perform multiplication directly between Q_S, K_S, V_S, which is then converted to sparse addition, like spike-driven Conv and MLP (SDSA-3/4 in this work).

B.2 Energy Consumption of Meta-SpikeFormer

When evaluating algorithms, the SNN field often ignores specific hardware implementation details and estimates theoretical energy consumption for a model (Panda et al., 2020; Yin et al., 2021; Yang et al., 2022; Yao et al., 2023d; Wang et al., 2023a). This theoretical estimation is just to facilitate the qualitative energy analysis of various SNN and ANN algorithms.

Theoretical energy consumption estimation can be performed in a simple way. For example, the energy cost of ANNs is FLOPs times E_{MAC}, and the energy cost of SNNs is FLOPs times E_{AC} times network spiking firing rate. $E_{MAC} = 4.6pJ$ and $E_{AC} = 0.9pJ$ are the energy of a MAC and an AC, respectively, in 45nm technology (Horowitz, 2014).

There is also a more refined method of evaluating energy consumption for SNNs. We can count the spiking firing rate of each layer, and then the energy consumption of each layer is FLOPs times E_{AC} times the layer spiking firing rate. The nuance is that the network structure affects the number of additions triggered by a single spike. For example, the energy consumption of the same spike tensor differs when doing matrix multiplication with various convolution kernel sizes.

In this paper, we count the spiking firing rate of each layer, then estimate the energy cost. Specifically, the FLOPs of the n-th Conv layer in ANNs Molchanov et al. (2017) are:

$$FL_{Conv} = (k_n)^2 \cdot h_n \cdot w_n \cdot c_{n-1} \cdot c_n, \tag{21}$$

where k_n is the kernel size, (h_n, w_n) is the output feature map size, c_{n-1} and c_n are the input and output channel numbers, respectively. The FLOPs of the m-th MLP layer in ANNs are:

$$FL_{MLP} = i_m \cdot o_m, \tag{22}$$

where i_m and o_m are the input and output dimensions of the MLP layer, respectively.

For spike-driven Conv or MLP, we only need to consider additional timestep T and layer spiking firing rates. The power of spike-driven Conv and MLP are $E_{AC} \cdot T \cdot R_C \cdot FL_{Conv}$ and $E_{AC} \cdot T \cdot R_M \cdot FL_{MLP}$ respectively. R_C and R_M represent the layer spiking firing rate, defined as the proportion of non-zero elements in the spike tensor. For the SDSA modules in Fig. 3, the energy cost of the Rep-Conv part is consistent with spike-driven Conv. The energy cost of the SDSA operator part is given in Table 6. Combining Table 5, we observe that the $SDSA(\cdot)$ function itself does not consume much energy because the Q, K, and V matrices themselves are sparse. The evidence is that SDSA-1 saves about 0.6mJ of energy consumption compared to SDSA-3 (see Table 5). In order to give readers an intuitive feeling about the spiking firing rate, we give the detailed spiking firing rates of a Meta-SpikeFormer model in Table 11.

C DETAILED CONFIGURATIONS AND HYPER-PARAMETER OF META-SPIKEFORMER MODELS

C.1 IMAGENET-1K EXPERIMENTS

On ImageNet-1K classification benchmark, we employ three scales of Meta-SpikeFormer in Table 7 and utilize the hyper-parameters in Table 8 to train models in our paper.

Table 7: Configurations of different Meta-SpikeFormer models.

stage	# Tokens	Layer Specification			15M	31M	55M
1	$\frac{H}{2}\times\frac{W}{2}$	Downsampling		Conv	7x7 stride 2		
				Dim	32	48	64
		Conv-based SNN block	SepConv	DWConv	7x7 stride 1		
				MLP ratio	2		
			Channel Conv	Conv	3x3 stride 1		
				Conv ratio	4		
	$\frac{H}{4}\times\frac{W}{4}$	Downsampling		Conv	3x3 stride 2		
				Dim	64	96	128
		Conv-based SNN block	SepConv	DWConv	7x7 stride 1		
				MLP ratio	2		
			Channel Conv	Conv	3x3 stride 1		
				Conv ratio	4		
2	$\frac{H}{8}\times\frac{W}{8}$	Downsampling		Conv	3x3 stride 2		
				Dim	128	192	256
		Conv-based SNN block	SepConv	DWConv	7x7 stride 1		
				MLP ratio	2		
			Channel Conv	Conv	3x3 stride 1		
				Conv ratio	4		
			# Blocks		2		
3	$\frac{H}{16}\times\frac{W}{16}$	Downsampling		Conv	3x3 stride 2		
				Dim	256	384	512
		Transformer-based SNN block	SDSA	RepConv	3x3 stride 1		
			Channel MLP	MLP ratio	4		
			# Blocks		6		
4	$\frac{H}{16}\times\frac{W}{16}$	Downsampling		Conv	3x3 stride 1		
				Dim	360	480	640
		Transformer-based SNN block	SDSA	RepConv	3x3 stride 1		
			Channel MLP	MLP ratio	4		
			# Blocks		2		

C.2 COCO EXPERIMENTS

In this paper, we have used two methods to utilize Meta-SpikeFormer for object detection. We first exploit Meta-SpikeFormer as backbones for object detection, fine-tuning for 24 epochs after inserting the Mask R-CNN detector (He et al., 2017). The batch size is 12. The AdamW is employed with an initial learning rate of 1×10^{-4} that will decay in the polynomial decay schedule with a power of 0.9. Images are resized and cropped into 1333×800 for training and testing and maintain the ratio. Random horizontal flipping and resize with a ratio of 0.5 was applied for augmentation during

Table 8: Hyper-parameters for image classification on ImageNet-1K

Hyper-parameter	Directly Training	Finetune
Model size	15M/31M/55M	15M/31M/55M
Timestemp	1	4
Epochs	200	20
Resolution	224*224	
Batch size	1568	336
Optimizer	LAMB	
Base Learning rate	6e-4	2e-5
Learning rate decay	Cosine	
Warmup eopchs	10	2
Weight decay	0.05	
Rand Augment	9/0.5	
Mixup	None	
Cutmix	None	
Label smoothing	0.1	

training. This pre-training fine-tuning method is a commonly used strategy in ANNs. We use this method and get SOTA results (see Table 3), but with many parameters. To address this problem, we then train Meta-SpikeFormer in a direct training manner in conjunction with the lightweight Yolov5 [1] detector, which Yolov5 is re-implemented by us in a spike-driven manner. Results are reported in Table 9. The current SOTA result in SNNs on COCO is EMS-Res-SNN (Su et al., 2023), which improves the structure. We get better performance using parameters that are close to EMS-Res-SNN.

Table 9: Performance of object detection on COCO val2017 (Lin et al., 2014)

Methods	Architecture	Spike -driven	Param (M)	Power (mJ)	Time Step	mAP@0.5 (%)
Conv-based SNN	EMS-Res-SNN (Su et al., 2023)	✓	26.9	-	4	50.1
Transformer-based SNN	Meta-SpikeFormer + Yolo (**This Work**)	✓	16.8	34.8	1	45.0
		✓	16.8	70.7	4	50.3

C.3 ADE20K EXPERIMENTS

Meta-SpikeFormer is employed as the backbone equipped with Sementic FPN Lin et al. (2017), which is re-implemented in a spike-driven manner. In $T = 1$, ImageNet-1K trained checkpoints are used to initialize the backbones while Xavier is utilized to initialize other newly added SNN layers. We train the model for 160K iterations with a batch size of 20. The AdamW is employed with an initial learning rate of 1×10^{-3} that will decay in the polynomial decay schedule with a power of 0.9. Then we finetuned the model to $T = 4$ and decreased the learning rate to 1×10^{-4}. To speed up training, we warm up the model for 1.5k iterations with a linear decay schedule.

C.4 ADDITIONAL RESULTS ON VOC2012 SEGMENTATION

VOC2012 (Everingham et al., 2010) is a benchmark for segmentation which has 1460 and 1456 images in the training and validation set respectively, and covering 21 categories. Previous work using SNN for segmentation has used this dataset. Thus we also test our method on this dataset. We train the Meta-SpikeFormer for 80k iterations in $T = 1$ with ImageNet-1k trained checkpoints to initialize the backbones while Xavier is utilized to initialize other newly added SNN layers. Then we finetune the model to $T = 4$ with lower learning rate 1×10^{-4}. Other experiment settings are the same as the ADE20k benchmark. Results are given in Table 10, and we achieve SOTA results.

[1] https://github.com/ultralytics/yolov5

Table 10: Performance of semantic segmentation on VOC2012 (Everingham et al., 2010)

Methods	Architecture	Spike -driven	Param (M)	Power (mJ)	Time Step	MIoU(%)
ANN	FCN-R50 (Long et al., 2015)	✗	49.5	909.6	1	62.2
	DeepLab-V3 (Chen et al., 2017)	✗	68.1	1240.6	1	66.7
ANN2SNN	Spike Calibration (Li et al., 2022)	✓	-	-	64	55.0
CNN-based SNN	Spiking FCN (Kim et al., 2022)	✓	49.5	383.5	20	9.9
	Spiking DeepLab (Kim et al., 2022)	✓	68.1	523.2	20	22.3
Transformer -based SNN	Meta-SpikeFormer **(This Work)**	✓	16.5	81.4	4	58.1
		✓	58.9	179.8	4	**61.1**

Table 11: Layer spiking firing rates of model Meta-SpikeFormer ($T=4$, 31.3M, SDSA-3) on ImageNet-1K.

				$T=1$	$T=2$	$T=3$	$T=4$	Average
Stage 1	Downsampling		Conv	1	1	1	1	1
	ConvBlock	SepConv	PWConv1	0.2662	0.4505	0.3231	0.4742	0.3785
			DWConv&PWConv2	0.3517	0.4134	0.3906	0.4057	0.3903
		Channel Conv	Conv1	0.3660	0.5830	0.4392	0.5529	0.4852
			Conv2	0.1601	0.1493	0.1662	0.1454	0.1552
	Downsampling		Conv	0.4408	0.4898	0.4929	0.4808	0.4761
	ConvBlock	SepConv	PWConv1	0.2237	0.3658	0.3272	0.3544	0.3178
			DWConv&PWConv2	0.2276	0.2672	0.2590	0.2567	0.2526
		Channel Conv	Conv1	0.3324	0.4640	0.4275	0.4433	0.4168
			Conv2	0.0866	0.0838	0.0811	0.0775	0.0823
Stage 2	Downsampling		Conv	0.3456	0.3916	0.3997	0.3916	0.3821
	ConvBlock	SepConv	PWConv1	0.2031	0.3845	0.3306	0.3648	0.3207
			DWConv&PWConv2	0.1860	0.2101	0.2020	0.1988	0.1992
		Channel Conv	Conv1	0.2871	0.4499	0.4013	0.4233	0.3904
			Conv2	0.0548	0.0541	0.0501	0.0464	0.0513
	ConvBlock	SepConv	PWConv1	0.3226	0.4245	0.4132	0.4158	0.3940
			DWConv&PWConv2	0.1051	0.1051	0.1025	0.0995	0.1030
		Channel Conv	Conv1	0.2863	0.3787	0.3732	0.3728	0.3528
			Conv2	0.0453	0.0418	0.0408	0.0382	0.0415
stage3	Downsampling		Conv	0.3817	0.4379	0.4436	0.4401	0.4259
	Block1	SDSA	RepConv-1/2/3	0.1193	0.2926	0.2396	0.2722	0.2309
			Q_S	0.2165	0.2402	0.2377	0.2213	0.2289
			K_S	0.0853	0.0931	0.0935	0.0818	0.0884
			V_S	0.0853	0.1414	0.1227	0.1234	0.1182
			$K_S^T V_S$	0.3083	0.4538	0.4238	0.4023	0.3971
			$Q_S(K_S^T V_S)$	0.7571	0.8832	0.8674	0.8426	0.8376
			RepConv-4	0.4115	0.6402	0.6034	0.5398	0.5487
		Channel MLP	Linear 1	0.2147	0.3849	0.3263	0.3637	0.3224
			Linear 2	0.0353	0.0298	0.0262	0.0232	0.0286
	Block2	SDSA	RepConv-1/2/3	0.2643	0.4093	0.3706	0.3918	0.3590
			Q_S	0.1594	0.1859	0.1913	0.1871	0.1809
			K_S	0.0774	0.1029	0.1061	0.1034	0.0975
			V_S	0.0852	0.1271	0.1228	0.1232	0.1146
			$K_S^T V_S$	0.4125	0.5852	0.5805	0.5835	0.5404
			$Q_S(K_S^T V_S)$	0.8246	0.9216	0.9231	0.9190	0.8970
			RepConv-4	0.4148	0.6622	0.6737	0.6545	0.6013
		Channel MLP	Linear 1	0.2899	0.4026	0.3756	0.3884	0.3641
			Linear 2	0.0302	0.0269	0.0239	0.0219	0.0258
	Block3	SDSA	RepConv-1/2/3	0.2894	0.3877	0.3706	0.3773	0.3562
			Q_S	0.1419	0.1397	0.1437	0.1405	0.1415
			K_S	0.0590	0.0609	0.0639	0.0616	0.0614
			V_S	0.0904	0.1232	0.1279	0.1261	0.1169
			$K_S^T V_S$	0.3674	0.4703	0.4825	0.4863	0.4516
			$Q_S(K_S^T V_S)$	0.8423	0.8912	0.9010	0.8961	0.8827
			RepConv-4	0.3613	0.4850	0.5281	0.5072	0.4704
		Channel MLP	Linear 1	0.3047	0.3795	0.3676	0.3727	0.3561
			Linear 2	0.0274	0.0248	0.0227	0.0211	0.0240

Continued on next page

Table 11 – continued from previous page

				$T=1$	$T=2$	$T=3$	$T=4$	Average
	Block4	SDSA	RepConv-1/2/3	0.2833	0.3469	0.3400	0.3430	0.3283
			Q_S	0.1910	0.1884	0.1937	0.1893	0.1906
			K_S	0.0570	0.0620	0.0658	0.0642	0.0622
			V_S	0.0834	0.0986	0.1065	0.1043	0.0982
			$K_S^T V_S$	0.3421	0.4375	0.4566	0.4670	0.4258
			$Q_S(K_S^T V_S)$	0.8279	0.8925	0.9067	0.9097	0.8842
			RepConv-4	0.3632	0.4932	0.5457	0.5365	0.4847
		Channel MLP	Linear 1	0.3040	0.3562	0.3487	0.3512	0.3400
			Linear 2	0.0282	0.0267	0.0244	0.0230	0.0256
	Block5	SDSA	RepConv-1/2/3	0.2882	0.3334	0.3280	0.3298	0.3198
			Q_S	0.1577	0.1487	0.1501	0.1482	0.1512
			K_S	0.0440	0.0496	0.0528	0.0534	0.0499
			V_S	0.0853	0.1276	0.1363	0.1377	0.1217
			$K_S^T V_S$	0.3633	0.4934	0.5187	0.5365	0.4780
			$Q_S(K_S^T V_S)$	0.8424	0.9031	0.9178	0.9213	0.8961
			RepConv-4	0.3550	0.5158	0.5620	0.5678	0.5001
		Channel MLP	Linear 1	0.3211	0.3551	0.3477	0.3503	0.3436
			Linear 2	0.0247	0.0223	0.0205	0.0194	0.0217
	Block6	SDSA	RepConv-1/2/3	0.3072	0.3335	0.3286	0.3310	0.3251
			Q_S	0.1468	0.1392	0.1392	0.1376	0.1407
			K_S	0.0373	0.0437	0.0442	0.0449	0.0426
			V_S	0.0935	0.1255	0.1331	0.1333	0.1213
			$K_S^T V_S$	0.3380	0.4449	0.4569	0.4667	0.4266
			$Q_S(K_S^T V_S)$	0.8073	0.8623	0.8706	0.8725	0.8532
			RepConv-4	0.2862	0.4085	0.4315	0.4352	0.3903
		Channel MLP	Linear 1	0.3084	0.3267	0.3192	0.3230	0.3193
			Linear 2	0.0241	0.0218	0.0202	0.0194	0.0214
stage4	Downsampling		Conv	0.2456	0.2487	0.2414	0.2438	0.2449
	Block1	SDSA	RepConv-1/2/3	0.1662	0.3402	0.3052	0.3280	0.2849
			Q_S	0.2044	0.1330	0.1202	0.1096	0.1418
			K_S	0.0221	0.0259	0.0214	0.0205	0.0225
			V_S	0.0870	0.1556	0.1438	0.1443	0.1327
			$K_S^T V_S$	0.1782	0.2832	0.2455	0.2412	0.2370
			$Q_S(K_S^T V_S)$	0.6046	0.6607	0.5710	0.5285	0.5912
			RepConv-4	0.2379	0.2635	0.1852	0.1592	0.2115
		Channel MLP	Linear 1	0.2332	0.3966	0.3615	0.3859	0.3443
			Linear 2	0.0262	0.0252	0.0192	0.0171	0.0219
	Block2	SDSA	RepConv-1/2/3	0.3053	0.4001	0.3907	0.4018	0.3745
			Q_S	0.1389	0.1245	0.1176	0.1108	0.1230
			K_S	0.0227	0.0231	0.0224	0.0218	0.0225
			V_S	0.0764	0.1038	0.1051	0.1048	0.0975
			$K_S^T V_S$	0.1600	0.1968	0.1985	0.1979	0.1883
			$Q_S(K_S^T V_S)$	0.5439	0.5558	0.5348	0.5079	0.5356
			RepConv-4	0.1718	0.1697	0.1578	0.1384	0.1594
		Channel MLP	Linear 1	0.3000	0.3811	0.3768	0.3913	0.3623
			Linear 2	0.0030	0.0035	0.0032	0.0029	0.0032
Head			Linear	0.4061	0.4205	0.4323	0.4545	0.4283