Agent Attention: On the Integration of Softmax and Linear Attention

Dongchen Han[1⋆], Tianzhu Ye[1⋆], Yizeng Han[1], Zhuofan Xia[1], Siyuan Pan[2], Pengfei Wan[2], Shiji Song[1], and Gao Huang[1⋆⋆]

[1] Department of Automation, Tsinghua University
[2] Kuaishou Technology

Appendix

A. Comparision with Related Works

Agent attention shares some similarities with two related works, namely GPViT [27] and GRL [12]. In this section, we provide a detailed analysis on their fundamental distinctions and the superiority of our work.

Firstly, agent attention's design is novel and superior. Agent attention introduces a set of agent tokens A to act as the "agent" for all queries Q, where A is usually directly acquired from the query space, i.e. $A=f(Q)$. In contrast, GRL [12] uses anchors A projected from the input X, i.e. $A=\text{Proj}(X)$, to compute cross-scale similarity (see its Fig. 6). GPViT [27] uses learnable tokens and additional MLPs to achieve global modeling (see its Fig. 3). As a result, agent attention can be applied to existing models in a training-free manner, *e.g.*, our AgentSD, which is impractical for GRL [12] and GPViT [27] since they need extra training of the projections or MLPs.

Secondly, our perspective of generalized linear attention is unique and essential. This perspective enables us to unleash the potential of agent attention with lightweight linear attention enhancements like DWC. In contrast, without such a view, GPViT [27] and GRL [12] have to compromise with other techniques, such as heavy MLP, window attention, or channel attention, to achieve comparable results.

Thirdly, agent attention is a superior alternative to Softmax attention, acting as a versatile module for general purposes, while GPViT [27] and GRL [12] are not plug-in modules and are limited to specific tasks.

B. Composition of Agent Bias

As mentioned in the main paper, to better utilize positional information, we present a carefully designed *Agent Bias* for our agent attention, i.e.,

$$O^{\text{A}}=\sigma(QA^T+B_2)\ \sigma(AK^T+B_1)\ V, \tag{1}$$

⋆ Equal contribution.
⋆⋆ Corresponding Author.

where $B_1 \in \mathbb{R}^{n\times N}, B_2 \in \mathbb{R}^{N\times n}$ are our agent biases. For parameter efficiency, we construct each agent bias using three bias components rather than directly setting B_1, B_2 as learnable parameters. For instance, values in B_1 are derived from column bias $B_{1c} \in \mathbb{R}^{n\times 1\times w}$, row bias $B_{1r} \in \mathbb{R}^{n\times h\times 1}$ and block bias $B_{1b} \in \mathbb{R}^{n\times h_0\times w_0}$, where h, w are the height and width of feature map or attention window, while h_0, w_0 are predefined hyperparameters much smaller than h, w. During the computation of attention weights, B_{1c}, B_{1r}, B_{1b} are resized to $B'_{1c}, B'_{1r}, B'_{1b} \in \mathbb{R}^{n\times h\times w}$ by repeating or interpolating, and $B_1 = (B'_{1c} + B'_{1r} + B'_{1b})$.reshape$(n, N)$ is used as the full agent bias.

C. Agent Attention for Stable Diffusion

C.1. Adjustments

As discussed in the main paper, when producing agent tokens through token merging, our agent attention can be directly applied to the Stable Diffusion model without any extra training. However, we are unable to apply the agent bias and DWC without training. As a remedy, we make two simple adjustments to our agent attention. Moreover, we change our agent attention module from

$$O = \sigma(QA^T + B_2)\ \sigma(AK^T + B_1)\ V + \mathrm{DWC}(V), \tag{2}$$

to

$$O = \sigma(QA^T)\ \sigma(AK^T)\ V + kV, \tag{3}$$

where k is a predefined hyperparameter. On the other hand, compared to the original softmax attention, the two softmax attention operations of agent attention may result in smoother feature distribution without training. In the light of this, we slightly increase the scale used for the second Softmax attention, i.e., agent broadcast.

C.2. Experiment Details

To quantitatively compare AgentSD with Stable Diffusion and ToMeSD, we follow [2] and employ Stable Diffusion v1.5 to generate 2,000 512^2 images of ImageNet-1k [5] classes, featuring two images per class, using 50 PLMS [15] diffusion steps with a cfg scale [7] of 7.5. Subsequently, we calculate FID [10] scores between these 2,000 samples and 50,000 ImageNet-1k validation examples, employing [21]. To assess speed, we calculate the average generation time of all 2,000 samples on a single RTX4090 GPU.

Complete quantitative results are presented in Tab. 1. Compared to SD and ToMeSD, our AgentSD not only accelerates generation and reduces memory usage, but also significantly improves image generation quality.

Table 1: Quantitative Results of Stable Diffusion, ToMeSD and our AgentSD. GB/img is measured as the total memory usage change when increasing batch size by 1.

Method	r	FID	s/img	GB/img
SD [19]	0	28.84	2.62	3.13
ToMeSD [2]	0.1	28.64	2.40	2.55
	0.2	28.68	2.15	2.03
	0.3	28.82	1.90	2.09
	0.4	28.74	1.71	1.69
	0.5	29.01	1.53	1.47
AgentSD	0.1	27.79	1.97	1.77
	0.2	27.77	1.80	1.60
	0.3	28.03	1.65	2.05
	0.4	28.15	1.54	1.55
	0.5	28.42	1.42	1.21

Table 2: Ablation on factor k of Eq. (3).

k	0	0.025	0.075	0.15
FID	28.80	28.67	28.42	28.61

C.3. Ablation

We further ablate the adjustments we made when applying agent attention to Stable Diffusion. As evident in Tab. 2 and Tab. 3, both adjustments to agent attention enhance the quality of AgentSD generation. Tab. 4 demonstrates that applying agent attention in the early stages yields substantial performance enhancements.

C.4. AgentSD for finetuning

Our agent attention module is also applicable in finetuning scenarios. To verify this, we select subject-driven task as an example and apply agent attention to SD-based Dreambooth [20]. We experimentally find that finetuning enables the integration of the agent attention module into all diffusion generation steps, reaching **2.2x** acceleration in generation speed compared to the original Dreambooth without sacrificing image quality. Additionally, time and memory cost during finetuning can be reduced as well.

Task and baseline. The diffusion subject-driven generation task entails maintaining the appearance of a given subject while generating novel renditions of it in different contexts, e.g., generating a photo of your pet dog dancing. Dreambooth [20] effectively addresses this task by finetuning a pretrained text-to-image diffusion model, binding a unique identifier with the given subject. Novel images of the subject can then be synthesized with the unique identifier.

Applying agent attention to Dreambooth. As previously discussed, Dreambooth [20] involves an additional finetuning process. We explore two approaches

Table 3: Ablation on scale used for the second Softmax attention.

Scale	$d^{-0.5}$	$d^{-0.25}$	$d^{-0.15}$	$d^{-0.05}$
FID	28.86	28.64	28.42	28.60

Table 4: Ablation on how many diffusion steps to apply agent attention.

Steps	early 20%	early 40%	early 60%	early 80%
FID	28.58	28.42	28.83	29.77
s/img	1.50	1.42	1.39	1.34

to applying agent attention to Dreambooth: (1) applying it *only during generation* and (2) applying it during *both finetuning and generation*. The first method is the same as the AgentSD detailed in the main paper, where we commonly apply agent attention to the early 40% of generation steps, achieving around a 1.7x speedup (merging ratio $r = 0.4$). However, applying agent attention to more diffusion steps for further acceleration leads to a decrease in image details and quality, as shown in Tab. 4 and the penultimate line of Fig. 1. Conversely, adopting the second approach, where the agent attention module is applied to all steps in both finetuning and generation, results in a 2.2x speedup in generation without sacrificing performance. Additionally, both time and memory costs in finetuning are reduced by around 15%, enabling model finetuning with less than 12GB of GPU memory in approximately 7 minutes on a single RTX4090 GPU. The last row in Fig. 1 shows the results of this setting.

Dataset and experiment details. We adopt the dataset provided by Dreambooth [20], which comprises 30 subjects of 15 different classes. It features live subjects and objects captured in various conditions, environments, and angles. We employ pretrained Stable Diffusion v1.5 and apply agent attention to all diffusion generation steps. The merging ratio r is set to 0.4, k is set to 0.075 and the scale for the second softmax attention is set to $d^{-0.15}$. We finetune all models for 800 iterations with a learning rate of 1e-6, utilizing 8-bit AdamW [6] as the optimizer. We follow [20] and select *sks* as the unique identifier for all settings. Novel synthesized images are sampled using the DDIM [22] sampler with 100 generation steps on a single RTX4090 GPU.

Visualization and discussion. Synthesized subject-driven images are shown in Fig. 1. We make two key observations: (1) Dreambooth with agent attention applied during finetuning and generation equals or surpasses the baseline Dreambooth in terms of fidelity and editability, and (2) employing agent attention during finetuning further enhances the fidelity and detail quality of synthesized images, enabling us to apply agent attention to all diffusion steps for more speedup. For the first observation, the first column shows that our method ensures the synthesized dog's color aligns more consistently with input images compared to the original Dreambooth and maintains comparable editability. For the second observation, comparing the last two rows of the third column reveals that applying agent attention to all diffusion steps without finetuning yields a blurry image, whereas our method produces a clearer and sharper depiction of

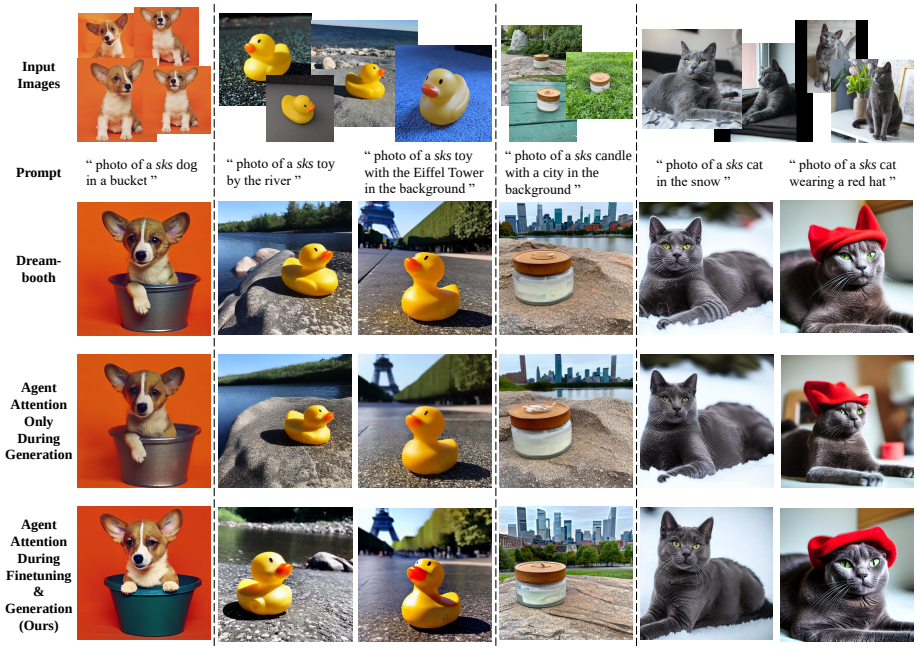

Fig. 1: Samples generated by Dreambooth and our Agent Dreambooth with the same seed. In the second-to-last line, we apply agent attention to all diffusion steps *only during generation*, leading to a slight decline in image quality as expected. In the last row, agent attention is incorporated into all steps in *both finetuning and generation*, resulting in a 2.2x speedup without compromising image quality. **Zoom in for best view.**

the duck toy. Additionally, in the fifth column, our method accurately generates the cat's eyes, whereas agent attention without finetuning fails in this aspect.

D. Dataset and Training Setup

D.1. ImageNet

Training settings. To ensure a fair comparison, we train our agent attention model with the same settings as the corresponding baseline model. Specifically, we employ AdamW [17] optimizer to train all our models from scratch for 300 epochs, using a cosine learning rate decay and 20 epochs of linear warm-up. We set the initial learning rate to 1×10^{-3} for a batch size of 1024 and linearly scale it *w.r.t.* the batch size. Following DeiT [23], we use RandAugment [4], Mixup [29], CutMix [28], and random erasing [30] to prevent overfitting. We also apply a weight decay of 0.05. To align with [8], we incorporate EMA [18] into the training of our Agent-CSwin models. For finetuning at larger resolutions, we follow the settings in [8, 16] and finetune the models for 30 epochs.

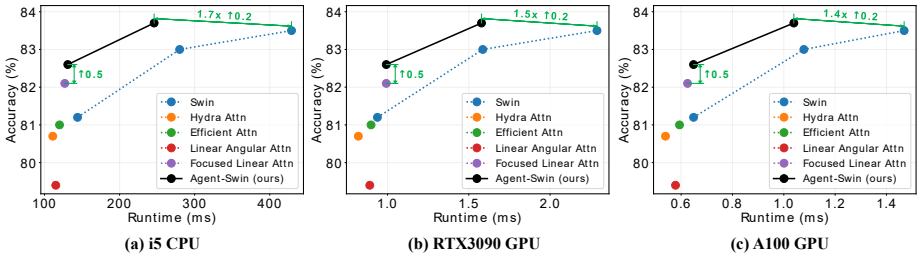

Fig. 2: Runtime comparison with other linear attention methods.

D.2. COCO

Training settings. COCO [14] object detection and instance segmentation dataset has 118K training and 5K validation images. We use a subset of 80k samples as training set and 35k for validation. Backbones are pretrained on ImageNet dataset with AdamW, following the training configurations mentioned in the original paper. Standard data augmentations including resize, random flip and normalize are applied. We set learning rate to 1e-4 and follow the 1x learning schedule: the whole network is trained for 12 epochs and the learning rate is divided by 10 at the 8th and 11th epoch respectively. For some models, we utilize 3x schedule: the network is trained for 36 epochs and the learning rate is divided by 10 at the 27th and 33rd epoch. All mAP results in the main paper are tested with input image size (3, 1333, 800).

Numbers of agent tokens. We use the ImageNet pretrained model as the backbone, which is trained with numbers of agent tokens n set to $[9, 16, 49, 49]$ for the four stages respectively. As dense prediction tasks involve higher-resolution images than ImageNet, we appropriately increase the numbers of agent tokens to better preserve the rich information. Specifically, for all the Agent-PVT models, we assign the numbers of agent tokens for the four stages as $[144, 256, 784, 784]$, while for all Agent-Swin models, we allocate $[81, 144, 196, 49]$. We employ bilinear interpolation to adapt the agent bias to the increased numbers of agent tokens n. The same strategy is applied to ADE20k experiments as well.

D.3. ADE20K

Training settings. ADE20K [31] is a well-established benchmark for semantic segmentation which encompasses 20K training images and 2K validation images. Backbones are pretrained on ImageNet with AdamW, following the training configurations mentioned in the original paper. For UperNet [26], we use AdamW to optimize, and set the initial learning rate as 6e-5 with a linear warmup of 1,500 iterations. Models are trained for 160K iterations in total. For Semantic FPN [11], we optimize the models using AdamW for 40k iterations with an initial learning rate of 2e-4. We randomly resize and crop the image to 512×512 for training, and re-scale to have a shorter side of 512 pixels during testing.

Table 5: Comparisons of agent attention with other vision transformer backbones on the ImageNet-1K classification task.

Method	Reso	#Params	Flops	Top-1
DeiT-T [23]	224^2	5.7M	1.2G	72.2
Agent-DeiT-T	224^2	6.0M	1.2G	**74.9 (+2.7)**
DeiT-S	224^2	22.1M	4.6G	79.8
Agent-DeiT-S	224^2	22.7M	4.4G	**80.5 (+0.7)**
DeiT-B	224^2	86.6M	17.6G	81.8
Agent-DeiT-B	224^2	87.2M	17.6G	**82.0 (+0.2)**
Agent-DeiT-S	448^2	23.1M	17.7G	**83.1 (+1.3)**
PVT-T [24]	224^2	13.2M	1.9G	75.1
Agent-PVT-T	224^2	11.6M	2.0G	**78.4 (+3.3)**
PVT-S	224^2	24.5M	3.8G	79.8
Agent-PVT-S	224^2	20.6M	4.0G	**82.2 (+2.4)**
PVT-M	224^2	44.2M	6.7G	81.2
Agent-PVT-M	224^2	35.9M	7.0G	**83.4 (+2.2)**
PVT-L	224^2	61.4M	9.8G	81.7
Agent-PVT-L	224^2	48.7M	10.4G	**83.7 (+2.0)**
Agent-PVT-M	256^2	36.1M	9.2G	**83.8 (+2.1)**
Swin-T [16]	224^2	29M	4.5G	81.3
Agent-Swin-T	224^2	29M	4.5G	**82.6 (+1.3)**
Swin-S	224^2	50M	8.7G	83.0
Agent-Swin-S	224^2	50M	8.7G	**83.7 (+0.7)**
Swin-B	224^2	88M	15.4G	83.5
Agent-Swin-B	224^2	88M	15.4G	**84.0 (+0.5)**
Agent-Swin-S	288^2	50M	14.6G	**84.1 (+0.6)**
Swin-B	384^2	88M	47.0G	84.5
Agent-Swin-B	384^2	88M	46.3G	**84.9 (+0.4)**
CSwin-T [8]	224^2	23M	4.3G	82.7
Agent-CSwin-T	224^2	21M	4.3G	**83.1 (+0.4)**
CSwin-S	224^2	35M	6.9G	83.6
Agent-CSwin-S	224^2	33M	6.8G	**83.9 (+0.3)**
CSwin-B	224^2	78M	15.0G	84.2
Agent-CSwin-B	224^2	73M	14.9G	**84.7 (+0.5)**
CSwin-B	384^2	78M	47.0G	85.4
Agent-CSwin-B	384^2	73M	46.3G	**85.8 (+0.4)**

Table 6: Results on COCO dataset. The FLOPs are computed over backbone, FPN and detection head with input resolution of 1280×800.

(a) Mask R-CNN Object Detection on COCO

Method	FLOPs	Sch.	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}
PVT-T	240G	1x	36.7	59.2	39.3	35.1	56.7	37.3
Agent-PVT-T	230G	1x	41.4	64.1	45.2	38.7	61.3	41.6
PVT-S	305G	1x	40.4	62.9	43.8	37.8	60.1	40.3
Agent-PVT-S	293G	1x	44.5	67.0	49.1	41.2	64.4	44.5
PVT-M	392G	1x	42.0	64.4	45.6	39.0	61.6	42.1
Agent-PVT-M	400G	1x	45.9	67.8	50.4	42.0	65.0	45.4
PVT-L	494G	1x	42.9	65.0	46.6	39.5	61.9	42.5
Agent-PVT-L	510G	1x	46.9	69.2	51.4	42.8	66.2	46.2
Swin-T	267G	1x	43.7	66.6	47.7	39.8	63.3	42.7
Agent-Swin-T	276G	1x	44.6	67.5	48.7	40.7	64.4	43.4
Swin-T	267G	3x	46.0	68.1	50.3	41.6	65.1	44.9
Agent-Swin-T	276G	3x	47.3	69.5	51.9	42.7	66.4	46.2
Swin-S	358G	1x	45.7	67.9	50.4	41.1	64.9	44.2
Agent-Swin-S	364G	1x	47.2	69.6	52.3	42.7	66.6	45.8
Swin-S	358G	3x	48.5	70.2	53.5	43.3	67.3	46.6
Agent-Swin-S	364G	3x	48.9	70.9	53.6	43.8	67.9	47.3

(b) Cascade Mask R-CNN Object Detection on COCO

Method	FLOPs	Sch.	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}
Swin-T	745G	1x	48.1	67.1	52.2	41.7	64.4	45.0
Agent-Swin-T	755G	1x	49.2	68.6	53.2	42.7	65.6	45.9
Swin-T	745G	3x	50.4	69.2	54.7	43.7	66.6	47.3
Agent-Swin-T	755G	3x	51.4	70.2	55.9	44.5	67.6	48.4
Swin-S	837G	3x	51.9	70.7	56.3	45.0	68.2	48.8
Agent-Swin-S	843G	3x	52.6	71.3	57.1	45.5	68.9	49.2
Swin-B	981G	3x	51.9	70.5	56.4	45.0	68.1	48.9
Agent-Swin-B	990G	3x	52.6	71.1	57.1	45.3	68.6	49.2

Table 7: Results on COCO object detection with RetinaNet [13]. The FLOPs are computed over backbone, FPN, and detection head with an input resolution of 1280×800.

RetinaNet Object Detection on COCO (Sch. 1x)

Method	FLOPs	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
PVT-T	221G	36.7	56.9	38.9	22.6	38.8	50.0
Agent-PVT-T	211G	40.3	61.2	42.9	25.5	43.4	54.3
PVT-S	286G	38.7	59.3	40.8	21.2	41.6	54.4
Agent-PVT-S	274G	44.1	65.3	47.3	29.2	47.5	59.8
PVT-M	373G	41.9	63.1	44.3	25.0	44.9	57.6
Agent-PVT-M	382G	45.8	66.9	49.1	28.8	49.2	61.7
PVT-L	475G	42.6	63.7	45.4	25.8	46.0	58.4
Agent-PVT-L	492G	46.8	68.2	50.7	30.9	50.8	62.9

Table 8: Results of semantic segmentation. The FLOPs are computed over encoders and decoders with an input image at the resolution of 512×2048. S-FPN is short for SemanticFPN [11] model.

Semantic Segmentation on ADE20K					
Backbone	Method	FLOPs	#Params	mIoU	mAcc
PVT-T	S-FPN	158G	17M	36.57	46.72
Agent-PVT-T	S-FPN	147G	15M	**40.18**	51.76
PVT-S	S-FPN	225G	28M	41.95	53.02
Agent-PVT-S	S-FPN	211G	24M	**44.18**	56.17
PVT-M	S-FPN	315G	48M	42.91	53.80
Agent-PVT-M	S-FPN	321G	40M	**44.30**	56.42
PVT-L	S-FPN	420G	65M	43.49	54.62
Agent-PVT-L	S-FPN	434G	52M	**46.52**	58.50
Swin-T	UperNet	945G	60M	44.51	55.61
Agent-Swin-T	UperNet	954G	61M	**46.68**	58.53
Swin-S	UperNet	1038G	81M	47.64	58.78
Agent-Swin-S	UperNet	1043G	81M	**48.08**	59.78
Swin-B	UperNet	1188G	121M	48.13	59.13
Agent-Swin-B	UperNet	1196G	121M	**48.73**	60.01

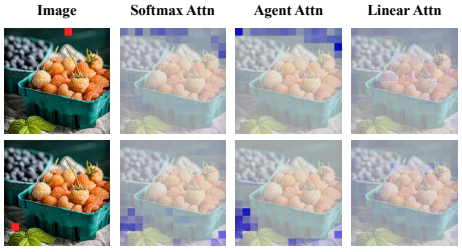

Fig. 3: Visualization of Softmax attention, linear attention and our agent attention. Feature corresponding to the red block is used as query.

E. Complete Experimental Results

Full classification results. We provide the full ImageNet-1K classification results (including high-resolution results) in Tab. 5. It is obvious that substituting Softmax attention with our agent attention in various models results in consistent performance improvements. We further provide additional runtime comparison with other linear attention methods in Fig. 2. Agent attention achieves superior results at a comparable speed compared to other linear attention methods.

Additional downstream experiments. We provide additional experiment results on object detection and semantic segmentation in Tab.7, Tab.6 and Tab.8. For object detection, results on RetinaNet [13], Mask R-CNN [9] and Cascade Mask R-CNN [3] frameworks are presented, while for semantic segmentation, we show results on SemanticFPN [11] and UperNet [26]. It can be observed that our

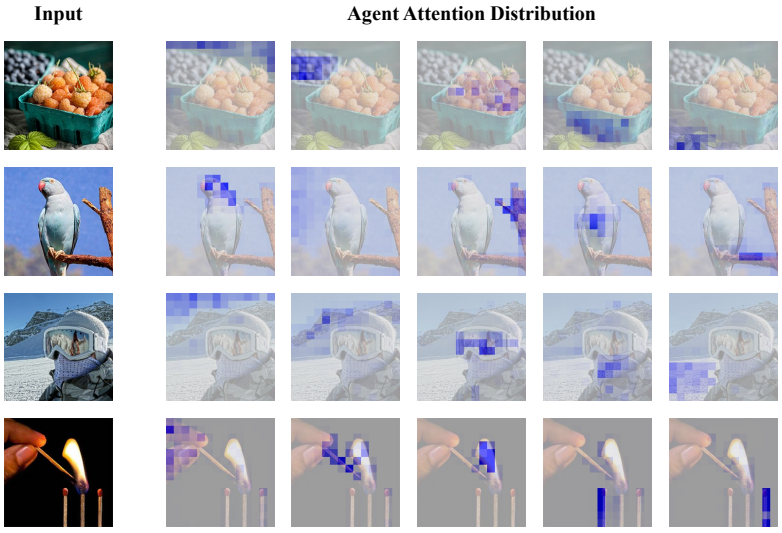

Fig. 4: The distribution of agent attention weights from Agent-Deit-T.

models achieve consistent improvements over their baseline counterparts across various settings.

Ablation on the type of agent tokens. Agent tokens can be acquired through various methods, such as setting as a set of learnable parameters or extracting from input features through pooling, depthwise convolution, etc. As shown in Tab. 9, dynamic agents outperform static ones due to their input-dependent nature, allowing for a more accurate representation of current queries. Pooling is a simple yet effective way to acquire agent tokens dynamically.

Table 9: Ablation on different designs of agent tokens.

Designs	Type	FLOPs	#Param	Acc.	Diff.
Learnable Params	Static	4.5G	29M	82.2	-0.4
Pooling	Dynamic	4.5G	29M	**82.6**	**Ours**
DWC	Dynamic	4.5G	29M	82.6	+0.0
Deformed Points [25]	Dynamic	4.5G	29M	82.7	+0.1
Token Merging [1]	Dynamic	4.6G	29M	82.6	+0.0

Agent attention at different stages. We conduct ablation study on replacing Softmax attention with our agent attention at different stages. As depicted in Tab. 10, substituting the first three stages results in a performance gain of 1.3, while replacing the final stage marginally decreases overall accuracy. We attribute this outcome to the larger resolutions in the first three stages, which are more conducive to agent attention module with a global receptive field.

Table 10: Applying agent attention module on different stages of the Swin-T structure.

Stages w/ Agent Attn				FLOPS	#Param	Acc.	Diff.
Stage1	Stage2	Stage3	Stage4				
✓				4.5G	29M	81.7	-0.9
✓	✓			4.5G	29M	81.8	-0.8
✓	✓	✓		4.5G	29M	**82.6**	**Ours**
✓	✓	✓	✓	4.5G	29M	82.5	-0.1
Swin-T				4.5G	29M	81.3	-1.3

F. Agent Attention Visualization

To better understand the effectiveness of agent attention, we provide the visualization of Softmax attention, linear attention and agent attention in Fig. 3. It shows that our agent attention produces attention distributions similar to Softmax attention, while linear attention does not generate reasonable distributions. This indicates that agent attention integrates Softmax attention's expressiveness with linear complexity, resulting in its superiority.

We visualize more agent attention distributions in Fig. 4. It can be seen that various agent tokens focus on distinct regions. For instance, in the second row, agent tokens focus on head, sky, body, and branch, while the third row contains agent tokens focusing on sky, mountain, glasses, mask, and ground. This diversity ensures that different queries can focus on their areas of interest during the agent broadcast process.

G. Model Architectures

Table 11: Architectures of Agent-DeiT models.

stage	output	Agent-DeiT-T		Agent-DeiT-S		Agent-DeiT-B	
		Agent	DeiT Block	**Agent**	DeiT Block	**Agent**	DeiT Block
res1	14×14	$\begin{bmatrix} \text{win } 14\times14 \\ \text{dim } 192 \\ \text{head } 3 \\ \text{agent } 49 \end{bmatrix} \times 12$	None	$\begin{bmatrix} \text{win } 14\times14 \\ \text{dim } 384 \\ \text{head } 6 \\ \text{agent } 49 \end{bmatrix} \times 12$	None	$\begin{bmatrix} \text{win } 14\times14 \\ \text{dim } 768 \\ \text{head } 12 \\ \text{agent } 81 \end{bmatrix} \times 4$	$\begin{bmatrix} \text{win } 14\times14 \\ \text{dim } 768 \\ \text{head } 12 \end{bmatrix} \times 8$

We present the architectures of four Transformer models used in the main paper, including Agent-DeiT, Agent-PVT, Agent-Swin and Agent-CSwin in Tab.11-15. Considering the advantage of enlarged receptive field, we mainly replace Softmax attention blocks with our agent attention module at early stages of vision Transformer models.

References

1. Bolya, D., Fu, C.Y., Dai, X., Zhang, P., Feichtenhofer, C., Hoffman, J.: Token merging: Your ViT but faster. In: ICLR (2023)

Table 12: Architectures of Agent-PVT models (Part1).

stage	output	Agent-PVT-T		Agent-PVT-S	
		Agent	PVT Block	**Agent**	PVT Block
res1	56 × 56	Conv4×4, stride=4, 64, LN			
		$\begin{bmatrix}\text{win } 56\times56\\ \text{dim } 64\\ \text{head } 1\\ \text{agent } 9\end{bmatrix}\times2$	None	$\begin{bmatrix}\text{win } 56\times56\\ \text{dim } 64\\ \text{head } 1\\ \text{agent } 9\end{bmatrix}\times3$	None
res2	28 × 28	Conv2×2, stride=2, 128, LN			
		$\begin{bmatrix}\text{win } 28\times28\\ \text{dim } 128\\ \text{head } 2\\ \text{agent } 16\end{bmatrix}\times2$	None	$\begin{bmatrix}\text{win } 28\times28\\ \text{dim } 128\\ \text{head } 2\\ \text{agent } 16\end{bmatrix}\times3$	None
res3	14 × 14	Conv2×2, stride=2, 320, LN			
		$\begin{bmatrix}\text{win } 14\times14\\ \text{dim } 320\\ \text{head } 5\\ \text{agent } 49\end{bmatrix}\times2$	None	$\begin{bmatrix}\text{win } 14\times14\\ \text{dim } 320\\ \text{head } 5\\ \text{agent } 49\end{bmatrix}\times6$	None
res4	7 × 7	Conv2×2, stride=2, 512, LN			
		$\begin{bmatrix}\text{win } 7\times7\\ \text{dim } 512\\ \text{head } 8\\ \text{agent } 49\end{bmatrix}\times2$	None	$\begin{bmatrix}\text{win } 7\times7\\ \text{dim } 512\\ \text{head } 8\\ \text{agent } 49\end{bmatrix}\times3$	None

2. Bolya, D., Hoffman, J.: Token merging for fast stable diffusion. In: CVPRW (2023)
3. Cai, Z., Vasconcelos, N.: Cascade r-cnn: Delving into high quality object detection. In: CVPR (2018)
4. Cubuk, E.D., Zoph, B., Shlens, J., Le, Q.V.: Randaugment: Practical automated data augmentation with a reduced search space. In: CVPRW (2020)
5. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: CVPR (2009)
6. Dettmers, T., Lewis, M., Shleifer, S., Zettlemoyer, L.: 8-bit optimizers via block-wise quantization. In: ICLR (2022)
7. Dhariwal, P., Nichol, A.: Diffusion models beat gans on image synthesis. In: NeurIPS (2021)
8. Dong, X., Bao, J., Chen, D., Zhang, W., Yu, N., Yuan, L., Chen, D., Guo, B.: Cswin transformer: A general vision transformer backbone with cross-shaped windows. In: CVPR (2022)
9. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: ICCV (2017)
10. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. In: NeurIPS (2017)
11. Kirillov, A., Girshick, R., He, K., Dollár, P.: Panoptic feature pyramid networks. In: CVPR (2019)
12. Li, Y., Fan, Y., Xiang, X., Demandolx, D., Ranjan, R., Timofte, R., Van Gool, L.: Efficient and explicit modelling of image hierarchies for image restoration. In: CVPR (2023)
13. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: ICCV (2017)
14. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: ECCV (2014)

Table 13: Architectures of Agent-PVT models (Part2).

stage	output	Agent-PVT-M		Agent-PVT-L	
		Agent	PVT Block	**Agent**	PVT Block
res1	56 × 56	Conv4×4, stride=4, 64, LN			
		$\begin{bmatrix}\text{win } 56\times56\\ \text{dim } 64\\ \text{head } 1\\ \text{agent } 9\end{bmatrix}\times3$	None	$\begin{bmatrix}\text{win } 56\times56\\ \text{dim } 64\\ \text{head } 1\\ \text{agent } 9\end{bmatrix}\times3$	None
res2	28 × 28	Conv2×2, stride=2, 128, LN			
		$\begin{bmatrix}\text{win } 28\times28\\ \text{dim } 128\\ \text{head } 2\\ \text{agent } 16\end{bmatrix}\times3$	None	$\begin{bmatrix}\text{win } 28\times28\\ \text{dim } 128\\ \text{head } 2\\ \text{agent } 16\end{bmatrix}\times8$	None
res3	14 × 14	Conv2×2, stride=2, 320, LN			
		$\begin{bmatrix}\text{win } 14\times14\\ \text{dim } 320\\ \text{head } 5\\ \text{agent } 49\end{bmatrix}\times18$	None	$\begin{bmatrix}\text{win } 14\times14\\ \text{dim } 320\\ \text{head } 5\\ \text{agent } 49\end{bmatrix}\times27$	None
res4	7 × 7	Conv2×2, stride=2, 512, LN			
		$\begin{bmatrix}\text{win } 7\times7\\ \text{dim } 512\\ \text{head } 8\\ \text{agent } 49\end{bmatrix}\times3$	None	$\begin{bmatrix}\text{win } 7\times7\\ \text{dim } 512\\ \text{head } 8\\ \text{agent } 49\end{bmatrix}\times3$	None

15. Liu, L., Ren, Y., Lin, Z., Zhao, Z.: Pseudo numerical methods for diffusion models on manifolds. In: ICLR (2022)
16. Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. In: ICCV (2021)
17. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: ICLR (2018)
18. Polyak, B.T., Juditsky, A.B.: Acceleration of stochastic approximation by averaging. SIAM Journal on Control and Optimization (1992)
19. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models. In: CVPR (2022)
20. Ruiz, N., Li, Y., Jampani, V., Pritch, Y., Rubinstein, M., Aberman, K.: Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In: CVPR (2023)
21. Seitzer, M.: pytorch-fid: FID Score for PyTorch. `https://github.com/mseitzer/pytorch-fid` (August 2020), version 0.3.0
22. Song, J., Meng, C., Ermon, S.: Denoising diffusion implicit models. In: ICLR (2021)
23. Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., Jégou, H.: Training data-efficient image transformers & distillation through attention. In: ICML (2021)
24. Wang, W., Xie, E., Li, X., Fan, D.P., Song, K., Liang, D., Lu, T., Luo, P., Shao, L.: Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In: ICCV (2021)
25. Xia, Z., Pan, X., Song, S., Li, L.E., Huang, G.: Vision transformer with deformable attention. In: CVPR (2022)
26. Xiao, T., Liu, Y., Zhou, B., Jiang, Y., Sun, J.: Unified perceptual parsing for scene understanding. In: ECCV (2018)
27. Yang, C., Xu, J., De Mello, S., Crowley, E.J., Wang, X.: Gpvit: a high resolution non-hierarchical vision transformer with group propagation. In: ICLR (2023)

Table 14: Architectures of Agent-Swin models.

stage	output	Agent-Swin-T		Agent-Swin-S		Agent-Swin-B	
		Agent	Swin Block	**Agent**	Swin Block	**Agent**	Swin Block
res1	56 × 56	concat 4 × 4, 96, LN		concat 4 × 4, 96, LN		concat 4 × 4, 128, LN	
		[win 56×56, dim 96, head 3, agent 9] ×2	None	[win 56×56, dim 96, head 3, agent 9] ×2	None	[win 56×56, dim 128, head 3, agent 9] ×2	None
res2	28 × 28	concat 2 × 2, 192, LN		concat 2 × 2, 192, LN		concat 2 × 2, 256, LN	
		[win 28×28, dim 192, head 6, agent 16] ×2	None	[win 28×28, dim 192, head 6, agent 16] ×2	None	[win 28×28, dim 256, head 6, agent 16] ×2	None
res3	14 × 14	concat 2 × 2, 384, LN		concat 2 × 2, 384, LN		concat 2 × 2, 512, LN	
		None	[win 7×7, dim 384, head 12] ×6	None	[win 7×7, dim 384, head 12] ×18	[win 14×14, dim 512, head 12, agent 49] ×2	[win 7×7, dim 512, head 12] ×16
res4	7 × 7	concat 2 × 2, 768, LN		concat 2 × 2, 768, LN		concat 2 × 2, 1024, LN	
		None	[win 7×7, dim 768, head 24] ×2	None	[win 7×7, dim 768, head 24] ×2	None	[win 7×7, dim 1024, head 24] ×2

Table 15: Architectures of Agent-CSwin models.

stage	output	Agent-CSwin-T		Agent-CSwin-S		Agent-CSwin-B	
		Agent	CSwin Block	**Agent**	CSwin Block	**Agent**	CSwin Block
res1	56 × 56	Conv7×7, stride=4, 64, LN				Conv7×7, stride=4, 96, LN	
		[win 56×56, dim 64, head 2, agent 9] ×2	None	[win 56×56, dim 64, head 2, agent 9] ×3	None	[win 56×56, dim 96, head 4, agent 9] ×3	None
res2	28 × 28	Conv7×7, stride=2, 128, LN				Conv7×7, stride=2, 192, LN	
		[win 28×28, dim 128, head 4, agent 16] ×4	None	[win 28×28, dim 128, head 4, agent 16] ×6	None	[win 28×28, dim 192, head 8, agent 16] ×6	None
res3	14 × 14	Conv7×7, stride=2, 256, LN				Conv7×7, stride=2, 384, LN	
		None	[win 7×14, dim 256, head 8] ×18	None	[win 7×14, dim 256, head 8] ×29	None	[win 7×14, dim 384, head 16] ×29
res4	7 × 7	Conv7×7, stride=2, 512, LN				Conv7×7, stride=2, 768, LN	
		None	[win 7×7, dim 512, head 16] ×1	None	[win 7×7, dim 512, head 16] ×2	None	[win 7×7, dim 768, head 32] ×2

28. Yun, S., Han, D., Oh, S.J., Chun, S., Choe, J., Yoo, Y.: Cutmix: Regularization strategy to train strong classifiers with localizable features. In: ICCV (2019)
29. Zhang, H., Cisse, M., Dauphin, Y.N., Lopez-Paz, D.: mixup: Beyond empirical risk minimization. In: ICLR (2018)
30. Zhong, Z., Zheng, L., Kang, G., Li, S., Yang, Y.: Random erasing data augmentation. In: AAAI (2020)
31. Zhou, B., Zhao, H., Puig, X., Xiao, T., Fidler, S., Barriuso, A., Torralba, A.: Semantic understanding of scenes through the ade20k dataset. IJCV (2019)