Zero-TPrune: Zero-Shot Token Pruning through Leveraging of the Attention Graph in Pre-Trained Transformers

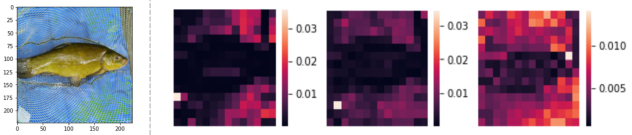
Supplementary Material

A. The I-S Pattern and the I′-S-I Pattern

In this section, we first demonstrate the *overwhelming of the major group* issue caused by the I-S pattern and then compare it with the I′-S-I pattern visually.

A.1. Overwhelming of the Major Group with the I-S Pattern

Sometimes, unimportant parts of an image may be identified as important and important parts as unimportant by our graph-based WPR algorithm. In the **I-stage**, each token votes for "more important tokens" and the weight of their votes is determined by their importance in the last round of voting. Besides the semantically significant tokens, tokens also intend to vote for tokens *that are similar to them*. When semantically significant tokens (e.g., main object tokens) are only a small part of an image and unimportant background tokens dominate, sometimes background tokens vote for each other and gradually obtain high importance scores after multiple rounds of voting. An example is shown in Fig. 1. It shows that the background of the image is considered important in the Transformer heads. The fish itself, surprisingly, is considered unimportant.



(a) (b)

Figure 1. An example illustrating that a large unimportant group may overwhelm a small important group: (a) input image and (b) three examples showing that unimportant background tokens overwhelm the important fish tokens.

In this image, the *background* and *fish* tokens form two sets: *A* and *B*. In the beginning, because tokens in *set B* are more semantically significant than those in *set A*, they have relatively high importance scores. However, both tokens in *set A* and *set B* mainly intend to vote for tokens in their own set. Thus, it is easier for *set A* to form tokens with high importance scores because *set A* includes more tokens. These "highly important" tokens have larger voting weights in the next iteration. This makes it even easier for other tokens in *set A* to get "high importance." This is a positive feedback loop, with the result that the most "important" tokens end up in *set A*.

A.2. Comparison

As shown in Fig. 2, by pruning similar background tokens in advance, the *overwhelming of the major group* problem is alleviated significantly.

B. Attention Probability Matrix

ViT [5] and its variants contain multiple Transformer encoder layers that are stacked up together. The basic Transformer encoder layer includes a multi-head attention (MHA) block followed by a feed-forward network (FFN) block, with residual connections and layer normalization around each. We make the assumption that an MHA block consists of H independently parameterized heads. An attention head h in layer l can be parameterized by the Key, Query, and Value weight matrices: $\mathbf{W}_k^{(h,l)}, \mathbf{W}_q^{(h,l)}, \mathbf{W}_v^{(h,l)} \in \mathbb{R}^{d_h \times d}$, and the output weight matrix $\mathbf{W}_o^{(h,l)} \in \mathbb{R}^{d \times d_h}$, where d_h is typically set to d/H and d is the embedded feature dimension. Suppose $x \in \mathbb{R}^{d \times n}$ is the input sequence and n is the input sequence length. For each head, the *attention probability* between token x_i and x_j is given as an element of matrix $\mathbf{A}^{(h,l)}$:

$$\mathbf{A}^{(h,l)}(x_i, x_j) = \text{softmax}\left(\frac{x^T \mathbf{W}_q^T \mathbf{W}_k x}{\sqrt{d}}\right)_{(i,j)} \in \mathbb{R} \quad (1)$$

This matrix measures how much token x_j attends to token x_i. The output of an MHA block can be formulated as follows:

$$x_{\text{MHA}} = \text{LN}\left(\mathbf{W}_o \sum_{i=1}^{n} \mathbf{W}_v x_i \mathbf{A}^{(h,l)}(x_i, x_j) + x\right) \quad (2)$$

The output of a Transformer encoder layer can be formulated as follows:

$$x_{\text{out}} = \text{LN}\left(\sigma\left(\mathbf{W}_2\left(\mathbf{W}_1 x_{\text{MHA}} + b_1\right)\right) + b_2 + x_{\text{MHA}}\right) \quad (3)$$

where $\mathbf{W}_1, \mathbf{W}_2, b_1$, and b_2 are FFN parameters, and σ and LN denote the activation function and layer normalization, respectively. We can see that the computation overhead of a Transformer encoder layer undergoes a quadratic reduction when tokens are pruned.

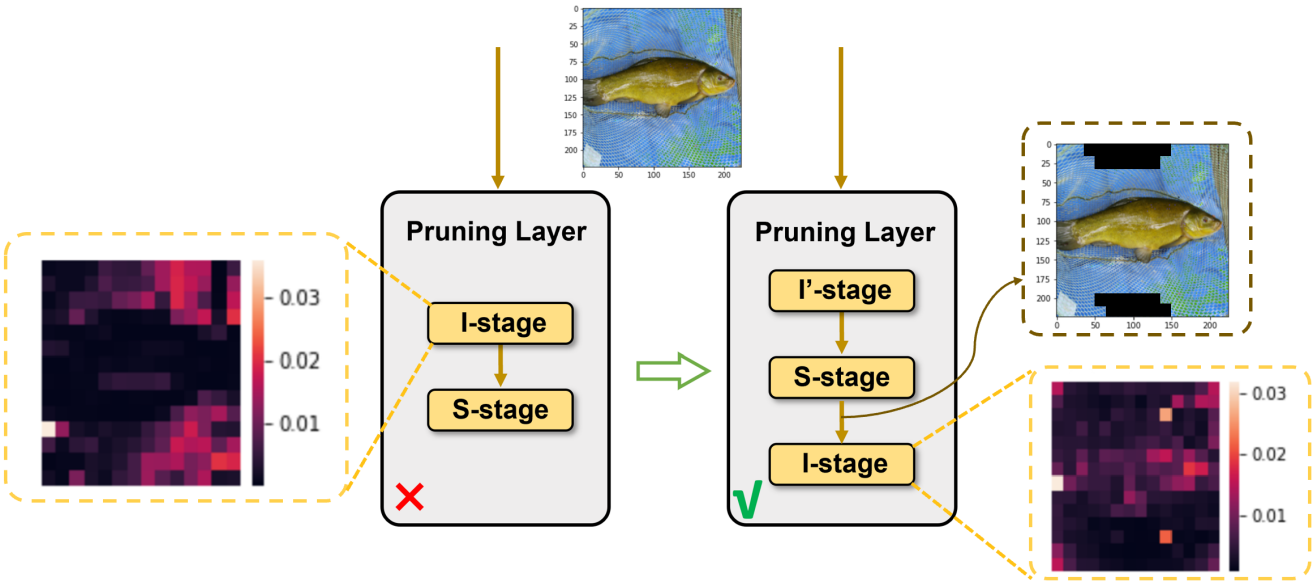


Figure 2. Visual comparison between the I-S pattern and the I′-S-I pattern.

C. Optional Training Paradigm after Pruning

Zero-TPrune can eliminate the fine-tuning process after pruning with a very small accuracy reduction. However, in some scenarios, we may have adequate samples and computational resources. In such cases, the performance of Zero-TPrune can be improved further by training (fine-tuning) after pruning. In this section, we introduce techniques used to accomplish this.

Given that it is very expensive to make importance-based ranking differentiable [4], we eliminate the **S-stage** and retain only the **I-stage** when we aim to further train (fine-tune) the pruned model. Besides this, to make Zero-TPrune differentiable, it is necessary to replace normal token pruning with "soft token pruning." Instead of completely discarding pruned tokens, soft token pruning assigns them small weights to reduce their effect on later computation and preserves compatibility with back-propagation during training. In this way, the non-differentiable token mask M is replaced with a differentiable soft mask $\tilde{M}$ using the sigmoid operation:

$$\tilde{M}^{(l)}\left(x_i\right) = \sigma\left(\frac{s^{(l)}\left(x_i\right) - \theta^{(l)}}{T}\right) \tag{4}$$

where $s^{(l)}\left(x_i\right)$ is the importance score of token x_i and $\theta^{(l)}$ is the importance threshold for the l-th layer. $\theta^{(l)}$ is determined based on the chosen pruning rate and GFLOPS budget. Details of soft token pruning can be found in [9].

For simplicity, we use a similar loss function to DynamicViT [16], which includes three terms:

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \lambda_{\text{distill}}\,\mathcal{L}_{\text{distill}} + \lambda_{\text{KL}}\mathcal{L}_{\text{KL}} \tag{5}$$

The first term is the standard classification cross-entropy loss:

$$\mathcal{L}_{\text{cls}} = \text{CrossEntropy}\left(\mathbf{y}, \overline{\mathbf{y}}\right) \tag{6}$$

During fine-tuning, we use the original backbone network as the teacher model and push the behavior of the Zero-TPrune model to be as close to the teacher model as possible. First, we push the finally retained tokens of Zero-TPrune close to the ones of the teacher model. This contributes to the second distillation term above. We also minimize the difference in predictions between Zero-TPrune and its teacher via Kullback-Liebler (KL) divergence. This contributes to the third term. Details of the loss function can be found in [16].

D. Visualization

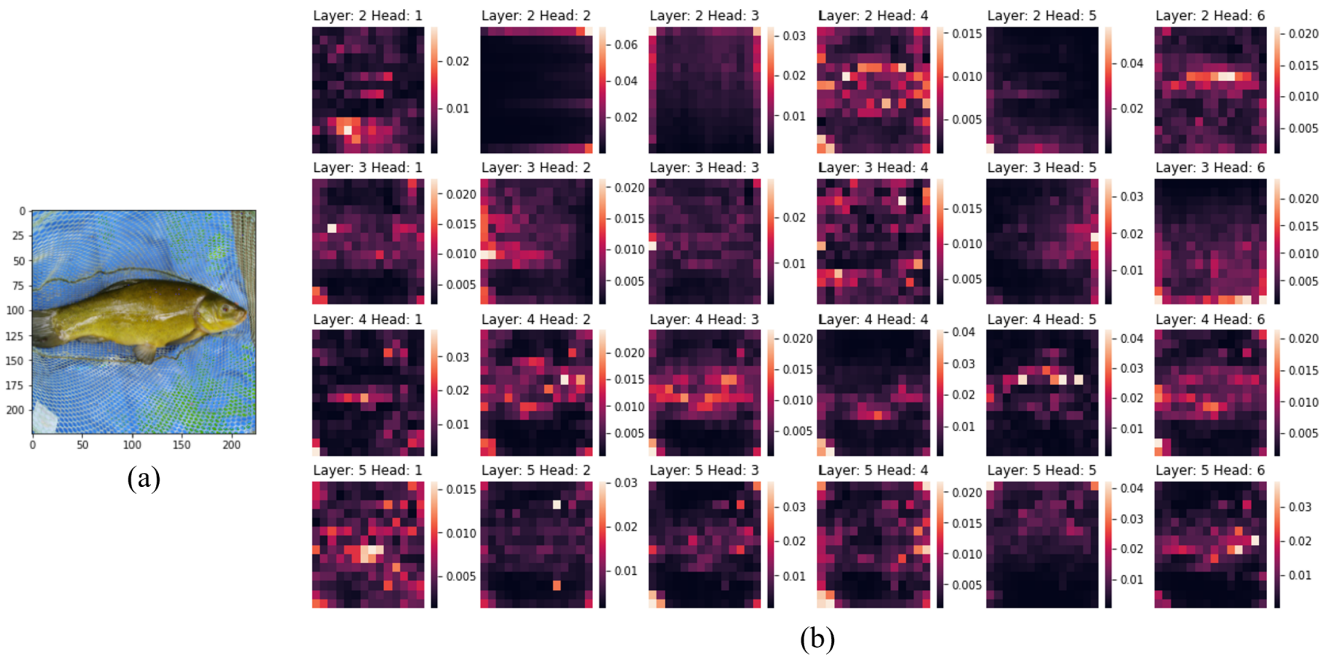
In this section, we use some visualization examples to provide high-level insights.

D.1. An Input Image Example

Fig. 3 shows a simple test sample of a *fish* from the ImageNet dataset and the corresponding importance score distributions in different layers and heads. We can see that most heads can successfully capture the important part of this image with the help of the graph-based WPR algorithm.

D.2. Averaged Importance Distribution over Thousands of Images

Another interesting visualization example is related to the general functionality of different layers in the Transformers.



(a) (b)

Figure 3. The important part of input images can be successfully captured by the graph-based WPR algorithm: (a) a test sample of *fish* in the ImageNet dataset and (b) the corresponding importance score distributions given by the WPR algorithm in different layers. The used backbone is DeiT-S.

Fig. 4 shows the importance score distributions averaged over thousands of images. It indicates that different layers of the Transformer behave differently. Shallow layers focus more on the edge of input images and deep layers focus more on the center.

E. Combining Results of Different Heads

In this section, we introduce the techniques we propose to nontrivially combine the importance score distribution of different heads from the WPR algorithm.

E.1. Emphasizing Informative Region

Different heads in an encoder layer usually pay attention to different parts of the input image, as shown in Fig. 5. For the input image of a boy holding a fish, some heads pay more attention to the body of this boy, some to the head of this boy, and some to the fish in hand.

We propose EIR to address this issue. Suppose there are three heads in all and the importance scores of tokens A, B, and C are [9,9,9], [9,0,0], [3,3,3], respectively. The ideal importance order is $A > B > C$. Table 1 shows the outcome of application of different importance score calculation methods. The traditional averaging method assigns the same importance to tokens A and B. If we only select the highest score across all heads, tokens A and B will be assigned the same importance, which is also not desired. The proposed EIP technique balances the two situations and results in the ideal importance order.

Table 1. Application of different importance score calculation methods to the example.

Importance Score	Average$\{S_i\}$	$max\{S_i\}$	EIP
Token A	9	9	5.2
Token B	3	9	3
Token C	3	3	1.7
Rank	$A > B = C$	$A = B > C$	$A > B > C$

E.2. Variance-based Head Filter

The importance scores given by the WPR algorithm may converge to an undesired distribution. Two typical examples are shown in Fig. 6. Tokens at the edge of the input image get very high importance scores in Fig. 6(b) and the importance score distribution in Fig. 6(c) is nearly uniform. We introduce VHF to mitigate the negative impact of these heads.

F. Downstream Tasks

Table 2 shows the number of categories and test instances in the selected datasets. DTD is a describable textures dataset; Indoor67 is an indoor scene recognition dataset; CUB200 is a challenging dataset of 200 bird species. The other datasets have self-explanatory names.

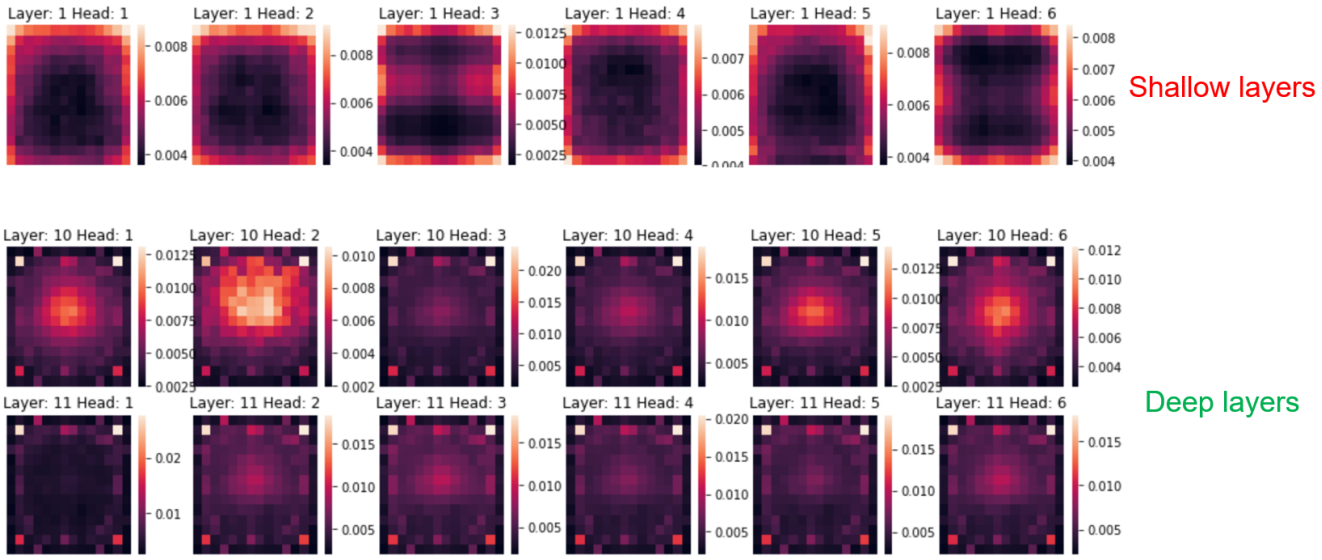


Figure 4. Importance score distributions averaged over thousands of images. The first row is derived from the first layer and the second (third) row from the 10th (11th) layer of the DeiT-S model.

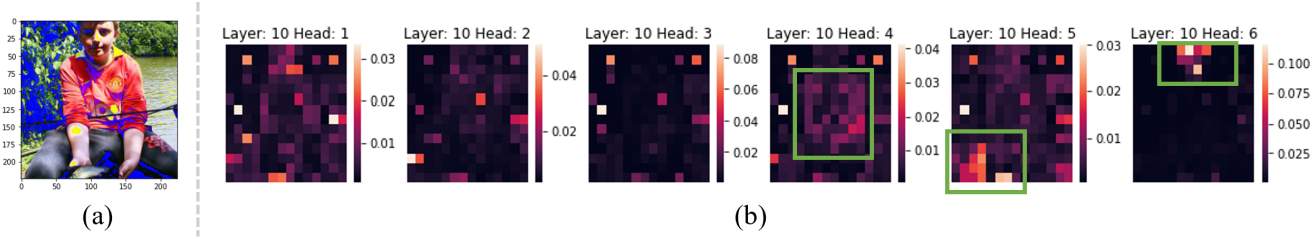


Figure 5. The distribution of importance score from different heads for an input image: (a) an image of a boy holding a fish and (b) importance score distributions. The results are obtained by the WPR algorithm with 30 iterations in the tenth layer of the DeiT-S model.

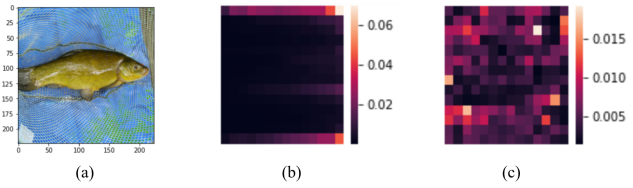


Figure 6. Examples of undesired importance score distributions in certain heads obtained by the WPR algorithm: (a) input image, (b) second head in the second layer of the DeiT-S model, and (c) fourth head in the third layer of the DeiT-S model.

Table 2. Datasets for downstream image classification.

Datasets	#Categories	#Test Instances
Flowers [13]	102	6149
Pets [14]	37	3669
DTD [3]	47	1880
Indoor67 [15]	67	1340
CUB200 [18]	200	5794
Aircrafts [12]	100	3333
Cars [10]	196	8041

The experimental results are shown in Table 3. ZeroTPrune outperforms baselines on most datasets, indicating its strong transfer learning capability after pruning. ToMe has worse performance on small-sized models due to a lack of enough layers to merge tokens gradually.

G. Ablation Experiments

In this section, we show results for further ablation experiments we performed. We explore the convergence speed of WPR and determine the appropriate number of iterations for each layer in Section G.1. Then we identify a good enough variance threshold for VHF in Section G.2. Furthermore, we describe optimal design choices in the **S-stage** in Sec-

Table 3. Performance of pruned models on downstream tasks.

Model	GFLOPS	Flowers	Pets	DTD	Indoor67	CUB200	Aircrafts	Cars
Deit-T	1.26	97.3	88.6	73.2	75.6	76.8	78.7	90.3
+ ATS	0.90	94.6	86.1	**71.0**	72.9	73.8	76.0	**88.4**
+ ToMe	0.90	93.2	84.7	69.9	71.6	72.9	75.2	87.1
+ Zero-TP	0.91	**95.1**	**86.9**	70.9	**73.7**	**74.4**	**76.7**	88.2

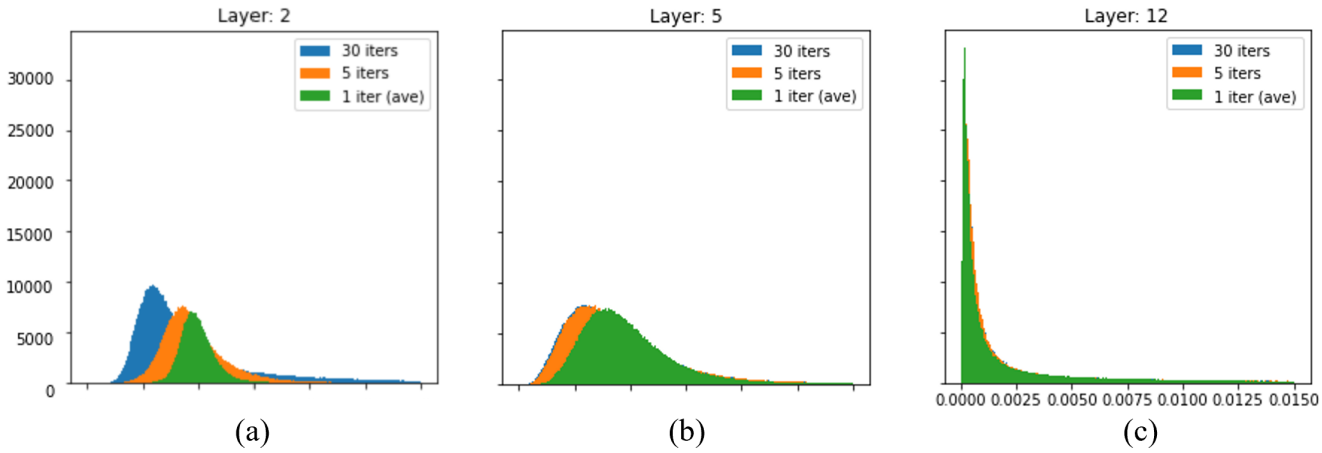


Figure 7. The importance score distributions of tokens in 2,560 images. The distribution changes with both the number of iterations and layer location: (a) layer 2, (b) layer 5, and (c) layer 12 in DeiT-S.

tion G.3, demonstrate the performance of Zero-TPrune-uni in Section G.4, and discuss hyperparameter search in Section G.5.

G.1. Convergence Speed of the WPR Algorithm

It is computationally expensive to check whether the WPR algorithm converges after each iteration. Thus, it would be desirable if we could determine the number of its iterations in advance. In order to do so, we need to derive the general convergence behavior of the WPR algorithm. Fig. 7 shows the importance score distributions of tokens in 2,560 images. In the shallow layers, such as the first layer, the distributions corresponding to 30 iterations and five iterations are obviously different. This indicates that five iterations are not enough to make the WPR algorithm converge in the shallow layers. On the other hand, in the deep layers, such as the 12th layer, the distribution corresponding to 30 iterations is quite similar to the distribution corresponding to just one iteration. This means that one iteration is enough to make the WPR algorithm converge in the deep layers. In addition, in the fifth layer, five iterations are enough to make it converge.

To quantitatively verify the assertions we made above, we calculate the KL divergence between the importance distribution given by 30, 5, 1 iteration(s) and that given by 50 iterations in different layers. The results are shown in Fig. 8. Thus, to ensure convergence, we set the number of iterations to 30-50, 5-10, and 1 in the first three layers, medium layers, and last three layers, respectively. Another interesting thing to note is that the Transformer model and the WPR algorithm assign low-importance scores to most tokens in the deep layers.

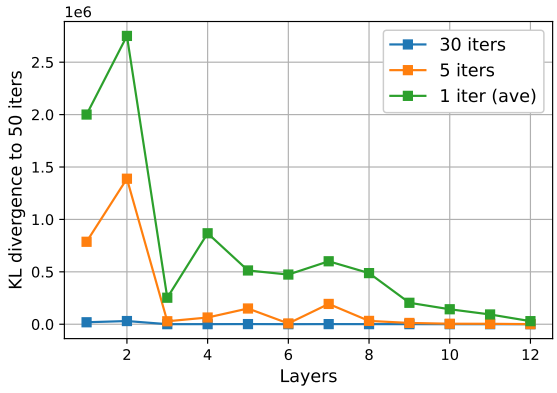


Figure 8. The KL divergence between the importance score distribution given by different numbers of iterations and that given by 50 iterations in different layers. The used backbone is DeiT-S.

G.2. Variance Thresholds for VHF

To exclude noise from heads that converge to undesired importance score distributions (as shown in Fig. 6), we propose VHF and set minimum and maximum thresholds for the variance of head distributions. We perform an ablation experiment to determine the optimal variance range. The pruning configuration is shown in Table 4. We then use random initialization and beam search ($k = 2$) to find a good enough variance range setting. The results are shown in Fig. 9, which points to the range [0.01,0.7].

Table 4. Pruning configuration used to search for optimal variance thresholds.

Pruning Layers	0	2	4	6	8	10
Retention Rates	0.9	0.9	0.85	0.8	0.7	0.65
# Iterations	50	50	5	5	1	1

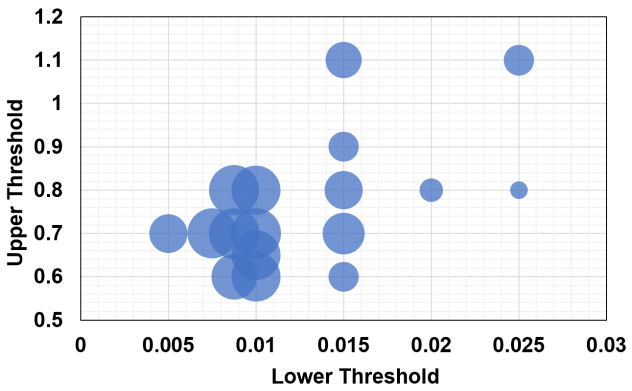


Figure 9. Results obtained in the process of searching for optimal thresholds. A larger blue bubble represents higher accuracy with that setting.

G.3. Optimal Design Choices in the S-stage

As discussed in Section 3.3 of our main paper, the design space of the **S-stage** is composed of three dimensions: (1) source of feature vectors, (2) partitioning method, and (3) similarity metric. We find that the optimal choice is (1) key matrix, (2) sequential (prune unimportant part), and (3) cosine similarity, respectively. This is the default setting in the following experiments unless otherwise noted. For the results in this section, pruning layers are inserted after the [1,3,6,9,11]-th layer with a retention rate of [1,0.9,0.8,0.7,1] and #iterations of [30,5,5,1,1] in the **I-stage**, and 10 tokens are pruned in each **S-stage**. Note that all results in this subsection are augmented by the CLS token by assigning it an importance score that is $\sqrt{N}$ times larger than other tokens during initialization in the **I-stage**, where N is the number of tokens.

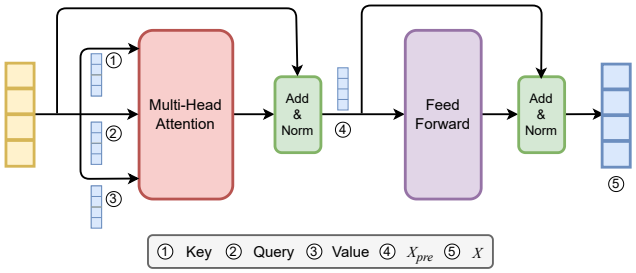


Figure 10. Potential feature vectors that can be used to represent tokens.

Table 5. Ablation experiment results for the source of **feature vectors**.

Feature	Acc@top1	GFLOPS
X_{pre}	79.113%	3.08
X	79.082%	3.08
K	**79.351%**	**3.08**
Q	79.205%	3.08
V	79.097%	3.08

Feature vectors: As shown in Fig. 10, feature vectors that represent tokens can be the corresponding vectors in the Key matrix, Query matrix, Value matrix, intermediate embedding vectors in the X_{pre} matrix, or output embedding vectors in the X matrix. We maintain the other settings and change the feature vectors used. The performance of pruned models is shown in Table 5. It indicates that the Key matrix is the optimal source of feature vectors.

Partitioning method: After ranking tokens according to their importance (e.g., *token* {*1,2,3,4,5,6*}; *token 1* has the highest score and *token 6* has the lowest), we choose from the following options: (i) **Alternate**: alternatively assign them to Group A and B, then the average token importance in two groups is nearly equal (e.g., $A : \{2, 4, 6\}$, $B : \{1, 3, 5\}$); (ii) **Sequential-U**: assign the less important half of tokens to Group A and the other half to Group B, which means we sequentially partition tokens and prune the unimportant part (e.g., $A : \{4, 5, 6\}$, $B : \{1, 2, 3\}$); (iii) **Sequential-I**: assign the more important half of tokens to Group A and the other half to Group B, which means we sequentially partition tokens and prune the important part (e.g., $A : \{1, 2, 3\}$, $B : \{4, 5, 6\}$), (iv) **Random**: randomly assign them to Group A or B; and (v) **No partition**: assign all tokens to both groups without partitioning. To evaluate the effectiveness of these options, we conducted experiments while keeping all other settings at default values. The results are shown in Table 6, where Sequential-U represents choice (ii) and Sequential-I represents choice (iii). It clearly indicates that Sequential-U is preferable to all the other partitioning methods.

Similarity metric: We experimented with several metrics for measuring similarity between two vectors, including cosine similarity, dot product, and Minkowski distance with different p values. When using Minkowski distance to measure similarity between vectors, we negated the distance to account for the fact that a longer distance indicates a lower similarity. The results of these experiments, shown in Table 7, indicate that cosine similarity is the best choice.

Table 6. Ablation experiment results for choosing the **partitioning method**.

Method	Acc@top1	GFLOPS
Random	79.055%	3.08
Alternate	79.179%	3.08
Sequential-U	**79.351%**	**3.08**
Sequential-I	78.898%	3.08
No partition	78.422%	3.08

Table 7. Ablation experiment results for choosing the **similarity metric**.

Similarity	Acc@top1	GFLOPS
dot product	79.257%	3.08
cosine	**79.351%**	**3.08**
Manhattan ($p = 1$)	79.208%	3.07
Euclidean ($p = 2$)	79.224%	3.07
Minkowski ($p = 3$)	79.246%	3.07
Minkowski ($p = 4$)	79.273%	3.07
Minkowski ($p = 5$)	79.189%	3.07
Minkowski ($p = \infty$)	79.092%	3.07

G.4. Performance of Zero-TPrune-uni

The ablation experimental results of Zero-TPrune-uni are shown in Table 8. The backbone for deployment is DeiT-S, and the model is evaluated on the ImageNet validation set.

G.5. Hyperparameter Search

The performance of Zero-TPrune, in terms of accuracy, is not sensitive to the hyperparameter setting as long as the number of pruning layers is more than two and the variance of their pruning rate is limited (i.e., the pruning process is not concentrated on one or two layers). We randomly choose different hyperparameter settings and show their performance in Fig. 11. This figure indicates that randomly selecting a hyperparameter setting does not hurt our performance much.

For a fair comparison with baselines, we do not use the best performance we can find through hyperparameter search. Instead, we use a hyperparameter setting with approximately the average performance among the search results. It is also close to setting a constant pruning rate across different layers.

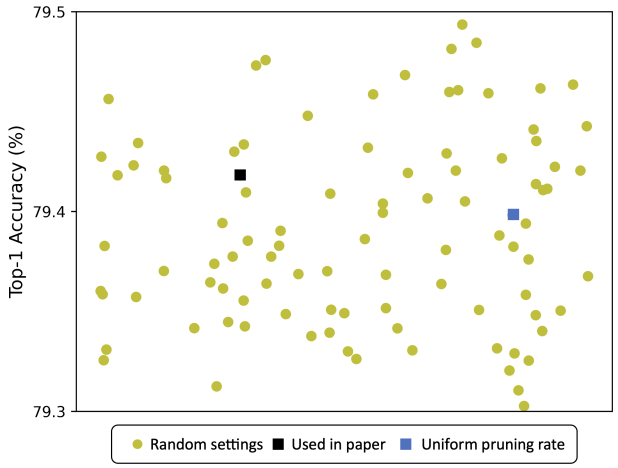


Figure 11. One hundred randomly selected hyperparameter settings and their corresponding performance after being applied to DeiT-S without fine-tuning

Even the full hyperparameter search process is much faster than fine-tuning. For hyperparameter search, we only need to perform inference. Specifically, for MCS, we randomly selected 1024 images in the validation dataset for each hyperparameter setting and obtain the corresponding accuracy. We tried 2000 settings on a single A100 GPU, which only required 3.8 hours. On the contrary, fine-tuning DeiT-S on the ImageNet dataset requires 144 A100 GPU hours.

H. Comparison with State-of-the-Art Methods

In this section, we first supplement comparisons with more depth-adaptive methods in Section H.1 and then compare Zero-TPrune with more straightforward attention-based token ranking methods in Section H.2. Finally, we provide performance comparisons with state-of-the-art fine-tuning-free token pruning methods in terms of throughput in Section H.3.

H.1. Depth-Adaptive Methods

Token pruning can be seen as a fine-grained variant of the depth-adaptive transformer, such as layer dropping. One of our baselines, A-ViT [19], is a token-wise depth-adaptive method. Instead of inserting pruning layers and setting pruning rates for them, it calculates the halting probability per token at each layer and halts tokens at adaptive depth. Zero-TPrune w/o fine-tuning competes with and even outperforms A-ViT w/ fine-tuning, as shown in Fig. 6 of our

Table 8. Contribution breakdown of the different techniques employed in Zero-TPrune-uni. The "-uni" suffix represents uniform initialization in the I-stage.

Acc@1	Params	GFLOPS	Throughput (img/s)	Method
79.8% (base)	22M	4.55G	1505.9	Unpruned model
76.8% (-3.0%)	22M	3.08G	2164.4	random drop
78.0% (+1.2%)	22M	3.08G	2142.3	WPR
78.2% (+0.2%)	22M	3.08G	2139.6	WPR + EIR
78.4% (+0.2%)	22M	3.08G	2107.2	WPR + EIR + VHF (I-stage)
78.9% (+0.5%)	22M	3.08G	2066.4	I-stage + S-stage
79.1% (+0.2%)	22M	3.08G	2062.9	I-stage + S-stage + MC Simulation

Table 9. Comparison with depth-adaptive methods on the DeiT-T model. The performance of Zero-TPrune is obtained without fine-tuning, while other results are obtained with fine-tuning.

Method	Acc@top1	GFLOPS
DeiT-T [17]	71.3%	1.3
ACT [8]	71.0%	1.0
Confidence threshold [11]	65.8%	1.1
Similarity gauging [6]	69.4%	1.1
PonderNet [1]	66.2%	1.0
DynamicViT [16]	70.9%	0.9
A-ViT [19]	71.0%	0.8
Zero-TPrune w/o FT	70.4%	0.9

main paper. A-ViT outperforms prior art on depth-adaptive methods. We adopt corresponding results and compare them with Zero-TPrune in Table 9. Note that the result of Zero-TPrune is obtained off the shelf without fine-tuning, while other results are obtained after fine-tuning the adaptive models.

H.2. Attention-based Token Ranking Methods

One of our baselines, ATS [7], is an attention-based importance ranking method. It uses the attention given by the CLS token to determine the importance of tokens. Simply averaging attention scores in the attention matrix is the baseline of ATS (Fig. 3 in [7]) and performs worse than ATS. For the ablation study, we replace our I-stage with top-k importance selection based on (1) CLS token attention, (2) average attention, and (3) accumulated average attention to improve the effectiveness of our method. Results are shown in Table 10. The batch size is 512 and our **S-stage** is enabled in all settings. We adjust pruning rates slightly to match the FLOPs cost of different settings. Our proposed **I-stage** uses information from all tokens while reducing noise from unimportant tokens, leading to better performance.

Table 10. Performance of pruned DeiT-S models without fine-tuning. Throughput is measured on a single NVIDIA A100 GPU.

Method	Acc@top1	GFLOPS	Throughput(img/s)
DeiT-S	79.8%	4.55	1505.9
CLS Attn.	78.9%	3.00	2179.3
Ave. Attn.	78.4%	2.99	2185.2
Accu. Ave. Attn.	78.5%	2.97	2189.2
I-stage	**79.4%**	2.97	2188.4

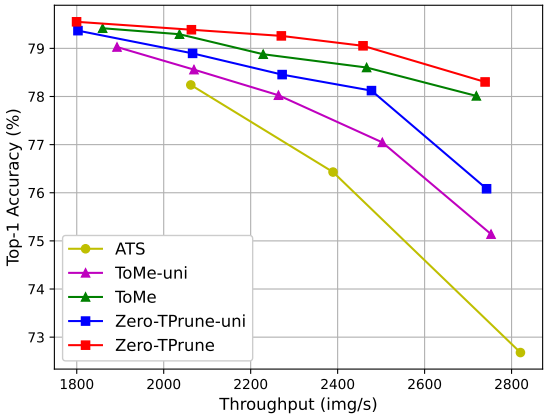


Figure 12. Performance comparison between Zero-TPrune and state-of-the-art fine-tuning-free methods. The applied Transformer backbone is DeiT-S.

H.3. Fine-Tuning-Free Token Pruning Methods

We conduct experiments on DeiT-S to show the superiority of Zero-TPrune over state-of-the-art fine-tuning-free token pruning/merging methods. The experimental results showing the trade-off between accuracy and throughput are shown in Fig. 12.

I. Comparison between Scaling and Pruning

As shown in Table 11, Zero-TPrune cannot outperform all baseline methods when a relatively high pruning rate (e.g.,

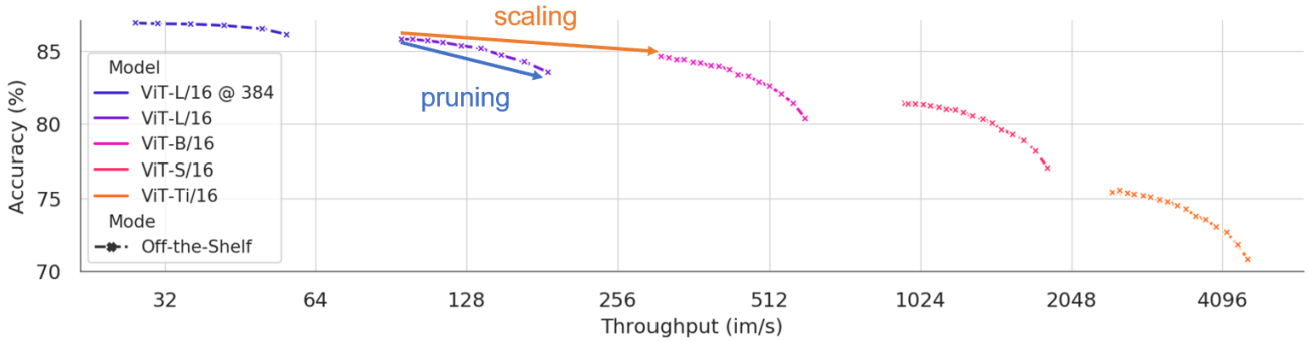


Figure 13. Off-the-shelf performance of ViT models under ToMe [2]. This figure is adopted from [2].

Table 11. Performance of pruned AugReg, LV-ViT, and SWAG models without fine-tuning. SWAG models perform inference on 384px images.

Method	Acc@top1	GFLOPS
LV-ViT-M	84.0%	12.7
+ ATS	80.9%	6.4
+ ToMe	**81.6%**	**6.3**
+ Zero-TP	81.4%	6.3
MAE	83.62%	55.4
+ATS	78.39%	29.1
+ToMe	**78.95%**	28.8
+Zero-TP	78.94%	28.6
SWAG	85.30%	55.6
+ATS	81.03%	27.8
+ToMe	**84.59%**	28.4
+Zero-TP	84.04%	28.3

reduce GFLOPS by 50%) is applied to large models (e.g., DeiT-L). However, in this case, scaling to a smaller model is often a better choice. ToMe outperforms Zero-TPrune when large models are aggressively pruned. Thus, we use the results from ToMe to illustrate this point.

In Fig. 13, ToMe is applied to different ViT backbones with different configurations. Different points on the same curve represent different configurations applied to the same backbone. The first point from the left on each curve represents the unpruned model. Aggressively pruning a model implies switching from the first point from the left on a given curve to the last point on this curve, which increases throughput but suffers from lower accuracy. Switching from the first point from the left on a given curve to the first point on another curve directly scales the size of the model without pruning. Aggressively pruning large models (ViT-L and ViT-B) underperforms scaling them in terms of both accuracy and throughput. On the contrary, for the ViT-S model, although scaling outperforms aggressive pruning in terms of throughput, it achieves lower accuracy than aggressive pruning.

References

[1] Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to Ponder. *arXiv preprint arXiv:2107.05407*, 2021.

[2] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token Merging: Your ViT But Faster. *arXiv preprint arXiv:2210.09461*, 2022.

[3] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing Textures in the Wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3606–3613, 2014.

[4] Marco Cuturi, Olivier Teboul, and Jean-Philippe Vert. Differentiable Ranking and Sorting Using Optimal Transport. *Advances in Neural Information Processing Systems*, 32, 2019.

[5] 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.

[6] Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-Adaptive Transformer. *arXiv preprint*, 2019.

[7] Mohsen Fayyaz, Soroush Abbasi Koohpayegani, Farnoush Rezaei Jafari, Sunando Sengupta, Hamid Reza Vaezi Joze, Eric Sommerlade, Hamed Pirsiavash, and Jürgen Gall. Adaptive Token Sampling for Efficient Vision Transformers. In *Proceedings of the European Conference on Computer Vision*, pages 396–414. Springer, 2022.

[8] Alex Graves. Adaptive Computation Time for Recurrent Neural Networks. *arXiv preprint arXiv:1603.08983*, 2016.

[9] Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Woosuk Kwon, Joseph Hassoun, and Kurt Keutzer. Learned Token Pruning for Transformers. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 784–794, 2022.

[10] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3D Object Representations for Fine-Grained Categorization. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 554–561, 2013.

[11] Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Haotang Deng, and Qi Ju. FastBERT: a Self-Distilling BERT with Adaptive Inference Time. *arXiv preprint arXiv:2004.02178*, 2020.

[12] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-Grained Visual Classification of Aircraft. *arXiv preprint arXiv:1306.5151*, 2013.

[13] Maria-Elena Nilsback and Andrew Zisserman. A Visual Vocabulary for Flower Classification. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1447–1454, 2006.

[14] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C.V. Jawahar. Cats and Dogs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3498–3505, 2012.

[15] Ariadna Quattoni and Antonio Torralba. Recognizing Indoor Scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 413–420, 2009.

[16] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. DynamicViT: Efficient Vision Transformers with Dynamic Token Sparsification. *Advances in Neural Information Processing Systems*, 34:13937–13949, 2021.

[17] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training Data-Efficient Image Transformers & Distillation through Attention. In *Proceedings of the International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

[18] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The Caltech-UCSD Birds-200-2011 Dataset. *California Institute of Technology, https://www.vision.caltech.edu/datasets/cub_200_2011/*, 2011.

[19] Hongxu Yin, Arash Vahdat, Jose M. Alvarez, Arun Mallya, Jan Kautz, and Pavlo Molchanov. A-ViT: Adaptive Tokens for Efficient Vision Transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10809–10818, 2022.