DiffRate : Differentiable Compression Rate for Efficient Vision Transformers

Mengzhao Chen[1,2,3†], Wenqi Shao[3*], Peng Xu[3,4], Mingbao Lin[5], Kaipeng Zhang[3], Fei Chao[1,2], Rongrong Ji[1,2*], Yu Qiao[3], Ping Luo[3,4]
[1]Key Laboratory of Multimedia Trusted Perception and Efficient Computing, Ministry of Education of China, Xiamen University
[2]Institute of Artificial Intelligence, Xiamen University
[3]OpenGVLab, Shanghai AI Laboratory [4]The University of Hong Kong [5]Tencent Holdings Ltd

Abstract

*Token compression aims to speed up large-scale vision transformers (e.g. ViTs) by pruning (dropping) or merging tokens. It is an important but challenging task. Although recent advanced approaches achieved great success, they need to carefully handcraft a compression rate (i.e. number of tokens to remove), which is tedious and leads to sub-optimal performance. To tackle this problem, we propose **Diff**erentiable Compression **Rate** (DiffRate), a novel token compression method that has several appealing properties prior arts do not have. First, DiffRate enables propagating the loss function's gradient onto the compression ratio, which is considered as a non-differentiable hyperparameter in previous work. In this case, different layers can automatically learn different compression rates layer-wisely without extra overhead. Second, token pruning and merging can be naturally performed simultaneously in DiffRate, while they were isolated in previous works. Third, extensive experiments demonstrate that DiffRate achieves state-of-the-art performance. For example, by applying the learned layer-wise compression rates to an off-the-shelf ViT-H (MAE) model, we achieve a 40% FLOPs reduction and a 1.5× throughput improvement, with a minor accuracy drop of 0.16% on ImageNet without fine-tuning, even outperforming previous methods with fine-tuning. Codes and models are available at https://github.com/OpenGVLab/DiffRate.*

1. Introduction

Vision Transformer (ViT) [7] has rapidly developed and achieved state-of-the-art performance in various vision tasks such as image classification[24], object detection [43], and semantic segmentation [34, 12]. Due to the flexibility in handling various input formats, ViT has also been widely applied to self-supervised learning [13] and other modalities [9, 35]. Despite the remarkable success, ViTs suffer from the intensive computational complexity that increases quadratically with the token length in the self-attention layer, presenting a challenge for practical applications. Therefore, it is crucial to improve the efficiency of ViTs in order to make them more widespread.

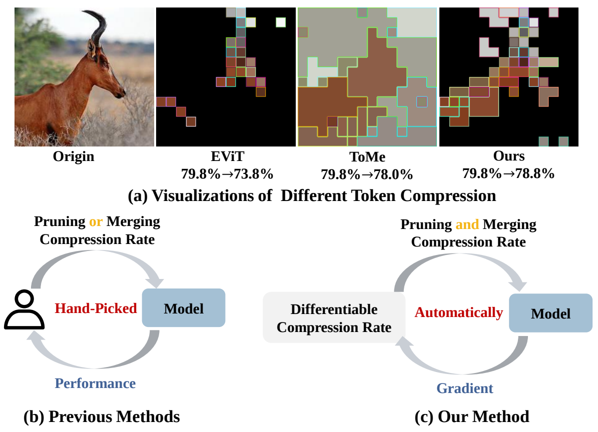


Figure 1: **Comparison of different token compression methods** including token pruning EViT [21], token merging ToMe [1] and our method. Pruned tokens are represented by black and non-border, while merged tokens are represented by patches with the same inner and border color. (**a**) shows that our method achieves better top-1 accuracy on ImageNet with FLOPs of 2.3G when compressing pre-trained Deit-S [30] without fine-tuning. (**b**) and (**c**) show that previous methods typically focus on either pruning or merging tokens using hand-picked compression rate with the guidance of performance. But our method leverages both approaches simultaneously to achieve more effective compression using the differentiable compression rate with gradient optimization.

In pursuit of efficient ViTs, various network compres-

*Corresponding authors: Rongrong Ji (rrji@xmu.edu.cn), Wenqi Shao (shaowenqi@pjlab.org.cn)
† This work was done during his internship at Shanghai AI Laboratory.

sion techniques such as weight pruning [37, 32], quantization [25, 40], distillation [15, 33] and so on have been investigated. Among them, token compression[27, 1, 21] has emerged as a promising approach to reduce redundancy in ViT for several appealing properties. First, token compression can be applied without any modifications to the network structure by exploiting the input-agnostic nature of transformers. Second, token compression is orthogonal to previous compression methods, making it a complementary approach to existing techniques [32, 4].

Existing token compression approaches typically include token pruning [27, 21, 18] and token merging [1]. As shown in Fig. 1(a), token pruning preserves informative tokens by measuring the importance of tokens with a defined metric. It can easily identify irrelevant background tokens with low importance. On the other hand, token merging compresses tokens by merging them with a large semantic similarity, which can not only discard some background tokens but also merge less informative foreground tokens. However, both pruning and merging handcraft a compression rate (*i.e.* the ratio of removed tokens and total tokens) for each transformer layer as shown in Fig. 1(b), which has two drawbacks. First, since model complexity metrics such as FLOPs are related to compression rates in each layer, it is tedious for practitioners to set layer-wise compression rates in order to meet the complexity constraints while retaining the performance of ViTs as much as possible. Second, informative foreground tokens are prone to being discarded with a hand-picked compression rate, which results in performance degradation. As shown in Fig. 1(a), when compressing tokens at fixed rates, token pruning such as EViT [21] removes most of the informative foreground tokens while token merging [1] also merges many important foreground tokens into a single token, leading to a sudden drop of top-1 accuracy on ImageNet.

To tackle the above issues, this work proposes a unified token compression framework called Differentiable Compression Rate (DiffRate), where both pruning and merging compression rates are determined in a differentiable manner. To achieve this goal, we propose a novel method, namely Differentiable Discrete Proxy (DDP) module. In DDP, a token sorting procedure is first performed to identify important tokens with a token importance metric. Then, a re-parameterization trick enables us to optimally select top-K important tokens with gradient back-propagation. In this way, all input images would have the top-K important tokens preserved, making it possible for parallel batch computation. Notably, the optimization process of DiffRate is highly efficient and can converge within 3 epochs (*i.e.* 2.7 GPU hours for ViT-B).

Thanks to the inclusion of differentiable compression rates, DiffRate can leverage the benefits of token pruning and merging by seamlessly integrating both techniques into a forward pass. This is possible because both token pruning and merging are capable of determining the optimal set of tokens to preserve. As shown in Fig. 1(a), DiffRate can prune most irrelevant background tokens and preserve detailed foreground information, leading to a good trade-off between efficiency and performance. With the learned compression rate, DiffRate achieves state-of-the-art performance in compressing various ViTs. For example, DiffRate can compress an off-the-shelf ViT-H model pre-trained by MAE [13] with 40% FLOPs reduction and 50% throughput improvement with only 0.16% accuracy drop, outperforming previous methods that require tuning the network parameter.

Our contributions are summarized as follows:

- We develop a unified token compression framework, Differentiable Compression Rate (DiffRate), that includes both token pruning and merging, and formulate token compression as an optimization problem.
- DiffRate employs a Differentiable Discrete Proxy which consists of a token sorting procedure and a re-parameterization trick to determine the optimal compression rate under different computation cost constraints. To our knowledge, it is the first study to explore differentiable compression rate optimization in token compression.
- Through extensive experiments, we demonstrate that DiffRate outperforms previous methods and achieves state-of-the-art performance on the off-the-shelf models. We hope that DiffRate can advance the field of token compression and improve the practical application of Vision Transformers (ViTs).

2. Related Work

Token Compression Several recent studies have attempt compress redundancy token according token pruning [21, 27, 8, 38, 29, 19, 18, 36, 22, 32] and token merging [1, 41, 28, 26]. However, most of these methods focus on designing metrics to distinguish redundant tokens, while ignoring the token compression schedule in each block. Some methods, such as VTC-LFC [32] and ViT-Slim [4] combine token prune with weight prune and determine the number of prune tokens using threshold-based approaches, which is also highly influenced by the hand-picked hyper-parameters. In contrast, our proposed method can learn the token compression schedule in a differentiable form. Furthermore, we consider both token merging and token pruning in the token compression process, resulting in a better trade-off between speed and accuracy.

Differentiable Neural Architecture Search. There are also many works [23, 3, 11] attempted to search for neural architecture in a differentiable manner. For example, DARTS [23] and ProxylessNAS [3] learn the probabilities of each candidate operation and select the operation with the highest probability as the final architecture. DMCP [11] is the approach most similar to proposed DiffRate, as it searches for the channel number of convolution in each layer. However, the differentiable method used in channel pruning cannot be directly applied to token compression. Firstly, the definition of search space is different. The search space for channel pruning is the channel, while for token compression, it is the token. Channels in the same position represent the same feature, while tokens are position-agnostic. Secondly, the output channel number in each layer is independent to other layers while the token must be pruned once it is discarded in previous layers. As of now, no approach has been proposed for differentiable token compression rate.

3. Differentiable Compression Rate

In this section, we first briefly introduce a transformer block and existing compression approaches, and then present our Differentiable Compression Rate (DiffRate) to build a unified token compression technique.

Transformer Block. Token compression in ViTs operates in each transformer block. Given the input token of the l-th block $\mathbf{X}^l \in \mathbb{R}^{N\times D}$ where N and D are the token length and token size, respectively, the forward propagation of the transformer block is expressed as follows:

$$\hat{\mathbf{X}}^l = \mathbf{X}^l + \mathrm{Attention}(\mathbf{X}^l), \mathbf{X}^{l+1} = \hat{\mathbf{X}}^l + \mathrm{MLP}(\hat{\mathbf{X}}^l), \quad (1)$$

where $l \in [L]$ and L is the network depth. Moreover, $\mathrm{Attention}$ and MLP represent the self-attention and the MLP modules in the transformer block, respectively. In Eqn. (1), $\hat{\mathbf{X}}^l$ is the output token of Attention.

Token Pruning and Merging. As shown in Fig. 2, existing token compression methods usually remove redundant tokens from $\hat{\mathbf{X}}^l$ by token pruning or merging, as given by

$$\hat{\mathbf{X}}^l_p \leftarrow f_p(\hat{\mathbf{X}}^l, \alpha^l_p) \text{ or } \hat{\mathbf{X}}^l_m \leftarrow f_m(\hat{\mathbf{X}}^l, \alpha^l_m), \quad (2)$$

where f_p, f_m are pruning and merging operations,α^l_p, α^l_m are their compression rates, and $\hat{\mathbf{X}}^l_p \in \mathbb{R}^{N^l_p\times D}, \hat{\mathbf{X}}^l_m \in \mathbb{R}^{N^l_m\times D}$ are their outputs which are then fed into MLP in Eqn. (1). Hence, the pruning and merging compression rate for each block is defined as $\alpha^l_p = (N - N^l_p)/N$ and $\alpha^l_m = (N - N^l_m)/N$, respectively. For example, EViT [21] preserves the important tokens while fusing unimportant tokens between Attention and MLP under the guidance of an importance metric. ToMe [1] merges similar tokens of $\hat{\mathbf{X}}^l$ in both foreground and background. Note that DynamicViT [27] prunes tokens after MLP, but we find that it also works well when it operates after Attention. Although these approaches achieved great success, they need to carefully

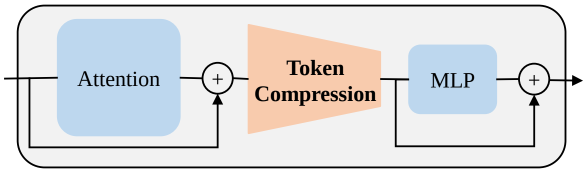


Figure 2: **Token compression** in a transformer block.

handcraft a compression rate block-wisely, which is tedious and leads to sub-optimal performance as shown in Fig. 1(a).

Unified Formulation of DiffRate. To mitigate this problem, we propose Differentiable Compression Rate (DiffRate), a unified token compression method, to search compression rates optimally. Given a pre-trained model $\mathbf{W}^*$, the objective of token compression is to minimize the classification loss $\mathcal{L}_{cls}$ on a training dataset $(\mathbf{X}, \mathbf{Y})$ with target FLOPs T. This can be formulated as an optimization problem as follows:

$$\boldsymbol{\alpha}^*_p, \boldsymbol{\alpha}^*_m = \arg\min_{\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_m} \mathcal{L}_{cls}(\mathbf{W}^*(\mathbf{X}), \mathbf{Y}|\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_m), \quad (3)$$

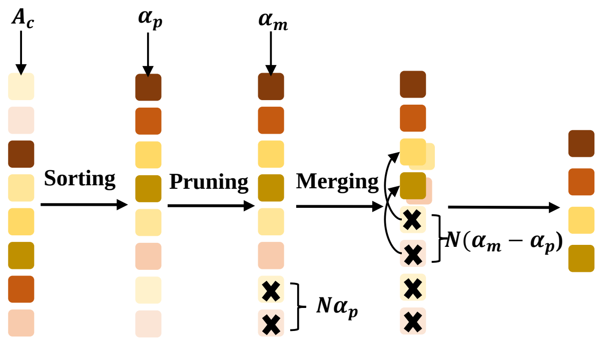
$$\text{s.t. } \mathcal{F}(\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_m) \le T, 0 \le \alpha^l_p, \alpha^l_m \le 1, \quad (4)$$

$$\hat{\mathbf{X}}^l = f_c(\hat{\mathbf{X}}^l, \alpha^l_p, \alpha^l_m), l \in [L] \quad (5)$$

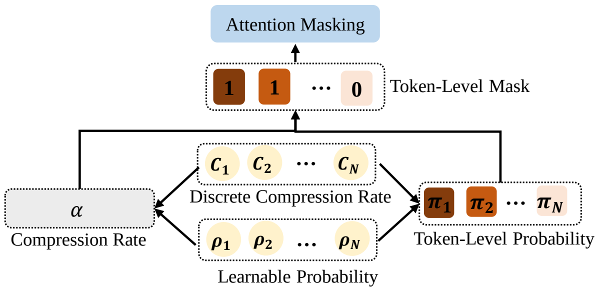
where $\boldsymbol{\alpha}_p = \{\alpha^l_p\}^L_{l=1}$ and $\boldsymbol{\alpha}_m = \{\alpha^l_m\}^L_{l=1}$ represent pruning and merging compression rates across all blocks, respectively. Moreover, $\mathcal{F}(\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_m)$ denotes the corresponding FLOPs, which can be expressed as a differentiable way of the compression rates. Eqn. (5) shows that DiffRate compresses $\hat{\mathbf{X}}^l$ with operation f_c and compression rates α^l_p and α^l_m in each transformer block. Finally, $\boldsymbol{\alpha}^*_p$ and $\boldsymbol{\alpha}^*_m$ are obtained by differentiably learning in DiffRate.

With the unified formulation of token compression, DiffRate is capable enough to express various compression methods. In detail, when $f_c = f_p, \alpha^l_m = 0$, DiffRate represents token pruning with differentiable pruning compression rate α^l_p. when $f_c = f_m, \alpha^l_p = 0$, DiffRate turn into differentiable token merging. In this work, we set $f_c = f_m \circ f_p$, meaning that tokens are first pruned and then merged. In this case, DiffRate seamlessly integrates token pruning and token merging through differentiable compression rates.

However, it is challenging to solve the optimization problem in Eqn. (3-5) with gradient-based methods for ViTs as compression rates are not differentiable. Directly learning 0-1 masks for tokens as did in channel pruning [17] is infeasible because each image may drop different numbers of tokens. This makes it hard to parallelize the computation. For example, DynamicViT [27] and SPViT [18] maintain a mask vector for each input image, but they still need to manually design compression rates to ensure that all images preserve the same number of tokens. The following section introduces a novel technique for the differentiable search of compression rates.



(a) Token Sorting



(b) Compression Rate Re-parameterization

Figure 3: **Pipeline of Differentiable Discrete Proxy**. (a) Token Sorting: the input N tokens are sorted based on the importance metric class attention A_c. With pruning rate α_p and merging rate α_m, we first prune $N\alpha^p$ least important tokens, then merge $N(\alpha^m - \alpha^p)$ unimportant tokens with similar ones among the remaining tokens. (b) Compression Rate Re-parameterization: the approach translate compression rate α into the combination with discrete rate $\mathbf{C}$ and learnable probability ρ. The top part is attention masking [27], which simulates token dropping by mask during training.

4. Differentiable Discrete Proxy

To make compression rates differentiable, our main idea is to preserve the top-K important tokens for all images, which can not only allow for parallel batch computation but also retain the performance of the original ViTs as much as possible. To achieve this, we introduce a novel method called the Differentiable Discrete Proxy (DDP), which comprises two critical components: a token sorting procedure to identify important tokens with a token importance metric and a re-parameterization trick to optimally select top-K important tokens with gradient back-propagation. The overall pipeline of DDP is illustrated in Fig. 3.

4.1. Token Sorting

Token Importance Metric. To find top-K importance tokens, we sort tokens by token importance metric, which has been well established in the literature. Here, we employ the class attention $\mathbf{A}_c \in \mathbb{R}^{1\times N}$ as the importance metric following EViT [21]. The interaction between class attention and image tokes can be written by:

$$\mathbf{A}_c = \text{Softmax}\big(\mathbf{q}_c\mathbf{K}^T/\sqrt{D}\big), \text{ and } \mathbf{X}_c = \mathbf{A}_c\mathbf{V}, \qquad (6)$$

where $\mathbf{q}_c \in \mathbb{R}^{1\times D}$, $\mathbf{K} \in \mathbb{R}^{N\times D}$, $\mathbf{V} \in \mathbb{R}^{N\times D}$ and $\mathbf{X}_c \in \mathbb{R}^{1\times D}$ denote the query vector of class token, the key matrix, the value matrix, and the class token of the self-attention layer. From Eqn. (6), the class attention $\mathbf{A}_c$ measures how much each image token contributes to the class token. Higher class attention indicates a more significant influence of the corresponding image token on the final output, implying greater importance [5, 21]. We also investigate other importance metrics in the ablation study as shown in Table 4a.

Pruning and Merging in DiffRate. With the token importance established in Eqn. (6), it is natural to remove tokens with low importance, such as those representing semantically irrelevant backgrounds by following principles in token pruning [21, 27]. As shown in Fig. 3a, we prune $N\alpha_p$ unimportant tokens in the l-th transformer block. Notely, we drop the superscript l of compression rate to simplify the notation. After that, we use cosine similarity to measure the similarity between $N(\alpha_m - \alpha_p)$ unimportant tokens and the remaining tokens. For similar token pairs, we generate a new token by directly average them. Through the above sorting-pruning-merging pipeline, the number of tokens to be pruned and merged in each block is optimally determined with learnable compression rate in our DiffRate. Hence, DiffRate can seamlessly integrate token pruning and merging.

4.2. Compression Rate Re-parameterization

DDP uses a re-parameterization trick to make pruning and merging compression rates differentiable. We simplify the notation by using a single variable α to represent both compression rates.

Re-parameterization with Discrete Rates. In essential, making compression rate differentiable is to determine how many tokens should be discarded with optimality guarantee. To tackle this problem, we re-parameterize the compression rate as a learnable combination of multiple candidate compression rates. Specifically, we introduce a discrete compression rate set, denoted as $\mathbf{C} = \{C_1, C_2, ..., C_N\}$, where $C_k = \frac{k-1}{N}$ represents the top $(k-1)$ least important tokens should be removed. By assigning learnable probabilities ρ_k to each candidate compression rate C_k with $\sum_{k=1}^{N} \rho_k = 1$, the compression rate can be written as

$$\alpha = \sum_{k=1}^{N} C_k\rho_k. \qquad (7)$$

By using discrete candidate rates as the proxy, the optimization problem of learning compression rates can be translated into the problem of learning the probabilities ρ_k.

Token-Level Mask. As shown in Fig. 3b, with C_k and ρ_k, the probability that the k-th important token is compressed can be calculated as

$$\pi_1 = 0, \pi_k = \rho_{N+2-k} + \cdots + \rho_{N-1} + \rho_N, k \geq 2 \quad (8)$$

where $\pi_1 = 0$ indicates that the most important token is always retained. From Eqn. (8), it is easy to see that $\pi_k \leq \pi_{k+1}$. Therefore, our DiffRate with DDP aligns with the fact that less important tokens should have a larger compression probability. To make the training and inference consistent, we convert π_k into a 0-1 mask as given by,

$$m_k = \begin{cases} 0, \pi_k \geq \alpha, \\ 1, \pi_k < \alpha, \end{cases} \quad (9)$$

where $m_k = 1$ indicates that the k-th token is preserved, and vice versa.

In each vision transformer block, we instantiate two independent re-parameterization modules to learn both pruning and merging compression rates. Thus, it generates two token-level masks, namely the pruning mask and the merging mask, denoted m_k^p and m_k^m for each token, respectively. Note that the token removed in last block must also be compressed in this block. Hence, the final mask is defined as

$$m_k = m_k \cdot m_k^p \cdot m_k^m, \quad (10)$$

where m_k on the right-hand side is the mask for the k-th token in the last block.

Attention Masking. To preserve the gradient back-propagation chain, we convert token dropping into attention masking with mask m_k in Eqn. (10) following DynamicViT [27]. To achieve this, we construct an attention mask $\mathbf{M}$ with the same dimensions as the attention map for each self-operation operation:

$$M_{i,j} = \begin{cases} 1, i = j, \\ m_j, i \neq j. \end{cases} \quad (11)$$

The attention mask prevents the interaction between all compressed tokens and the other tokens, except itself. We then use this mask to modified the $\mathrm{Softmax}$ operation in the next self-attention module:

$$\mathbf{S} = \frac{\mathbf{QK}^T}{\sqrt{D}}, \hat{S}_{i,j} = \frac{\exp(S_{i,j})M_{i,j}}{\sum_{k=1}^{N}\exp(\mathrm{S_{i,k}})M_{i,k}}, \quad (12)$$

where $\mathbf{Q} \in \mathbb{R}^{N\times D}$ is the query matrix, $\mathbf{S} \in \mathbb{R}^{N\times N}$ is the original attention map before $\mathrm{SoftMax}$, and $\hat{S}_{i,j}$ is actually used to update tokens. Eqn. (11-12) enables propagation the loss function's gradient onto the mask m.

Algorithm 1 Differentiable Compression Rate.

Input: training dataset $(\mathbf{X}, \mathbf{Y})$, pretrained model weight $\mathbf{W}^*$, target FLOPs T, DDP with discrete compression rate $\{C_k\}_{k=1}^N$ and learnable probabilities $\{\rho_k\}_{k=1}^N$.

Output: block-wise pruning compression rate $\boldsymbol{\alpha}_p = \{\alpha_p^l\}_{l=1}^L$ and merging compression rate $\boldsymbol{\alpha}_m = \{\alpha_m^l\}_{l=1}^L$.

```
1:  for $(\mathbf{x}, \mathbf{y})$ in $(\mathbf{X}, \mathbf{Y})$ do
2:      calculate $\boldsymbol{\alpha}_p$ and $\boldsymbol{\alpha}_m$ with $\{\rho_k\}_{k=1}^N$ by Eqn. (7).
3:      calculate pruning mask $\{m_k^p\}_{k=1}^N$ by Eqn. (9).
4:      calculate merging mask $\{m_k^m\}_{k=1}^N$ by Eqn. (9).
5:      sorting $\rightarrow$ pruning $\rightarrow$ merging in Sec. 4.1
6:      attention masking with Eqn. (11-12)
7:      calculate classification loss: $\mathcal{L}_{cls}(\mathbf{W}^*(\mathbf{x}), \mathbf{y})$
8:      calculate FLOPs loss: $L_f = (\mathcal{F}(\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_m) - T)^2$
9:      calculate optimization objective: $\mathcal{L} = \mathcal{L}_{cls} + \lambda_f \mathcal{L}_f$
10:     backward $\mathcal{L}$ to $\rho_k$ with Eqn. (14)
11:     update $\rho_k$ by gradient
12: end for
13: return $\boldsymbol{\alpha}_p$ and $\boldsymbol{\alpha}_m$
```

5. Training Objective

We solve the optimization problem in Eqn. (3-5) by minimizing the total loss as given by

$$\mathcal{L} = \mathcal{L}_{cls} + \lambda_f \mathcal{L}_f(\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_m), \quad (13)$$

where $\mathcal{L}_f = (\mathcal{F}(\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_m) - T)^2$ is the loss to constraint the FLOPs. The hyper-parameter λ_f balances the two loss terms, and we set it to 5 by default in our experiments.

During network back-propagation, we utilize the straight-through-estimator (STE) [14] to calculate the gradient of Eqn. (11). Hence, we can calculate the gradient of $\mathcal{L}$ with respect to ρ_k using the chain rule:

$$\frac{\partial \mathcal{L}}{\partial \rho_k} = \sum_{j=1}^{N} \frac{\partial \mathcal{L}}{\partial m_j} \frac{\partial m_j}{\partial \pi_j} \frac{\partial \pi_j}{\partial \rho_k} \approx \sum_{j=1}^{N} \frac{\partial \mathcal{L}}{\partial m_j} \frac{\partial \pi_j}{\partial \rho_k}. \quad (14)$$

Since ρ_k is differentiable through Eqn. (14), the compression rate α can be optimized with gradient back-propagation by Eqn. (7).

Overall Algorithm. The overall training algorithm of DiffRate is illustrated in Algorithm 1. It consists of three steps: forward model with ρ_k (Lines 2-6), calculating optimization objective (Lines 7-9), backward propagation and ρ_k update in DDP (Lines 10-11). The DiffRate algorithm finds the optimal compression rate by updating ρ_k in a differentiable form, and the resulting compression rate can be directly applied to off-the-shelf models.

Extension to Other Complexity Metrics. Our DiffRate model offers flexible differentiable compression rates that can be supervised using various computational complexity metrics, such as FLOPs and latency. To investigate its potential, we employed Gemmini [10], a framework for generating deep learning accelerators that can produce a diverse set of realistic accelerators based on a flexible architecture template. Using Gemmini, we conducted a co-search of the design space for compression ratios $\boldsymbol{\alpha}_p$ and $\boldsymbol{\alpha}_m$, as well as accelerator parameters $\boldsymbol{\beta}$ simultaneously. The loss function in Eqn. (13) becomes

$$\begin{aligned}\mathcal{L} &= \mathcal{L}_{cls} + \lambda_f \mathcal{L}_f(\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_m) \\ &+ \lambda_{la}\mathcal{L}_{la}(\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_m, \boldsymbol{\beta}) + \lambda_{pw}\mathcal{L}_{pw}(\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_m, \boldsymbol{\beta}),\end{aligned} \tag{15}$$

where $\mathcal{L}_{la}$ and $\mathcal{L}_{pw}$ are loss functions to constraint latency and power consumption in Gemmini accelerator, respectively. λ_{la} and λ_{pw} are their strengths. By minimizing $\mathcal{L}$, DiffRate can further learn optimal $\boldsymbol{\alpha}_p^*, \boldsymbol{\alpha}_m^*$ satisfying various resource constraints, and the optimum $\boldsymbol{\beta}^*$ can be implemented in FPGA board. The details are in Appendix B.

6. Experiments

This section presents extensive experiments to verify our proposed DiffRate. Sec. 6.1 provides training details. Sec. 6.2 compares DiffRate with previous compression techniques. The ablation study and visualization are presented in Sec. 6.3 and Sec. 6.4, repsectively.

6.1. Implementation Details

In this section, we conduct a series of experiments on ImageNet-1k [6] using DeiT [30], MAE [13], and LV-ViT [16]. We initialize the backbone models with pre-trained models and fix them to train the learnable probabilities in DDP with the objective function in Eqn. (13). The DDP is trained for 3 epochs using a learning rate of $1e^{-2}$. Once the compression rate of DDP is determined, it can be directly applied to off-the-shelf models. We also fine-tune the model optionally for 30 epochs using a learning rate of $1e^{-5}$ after determining the compression rate, denoted by the superscript "†" in the table for differentiation. All throughput measurements are taken during inference on an A100 GPU with a batch size of 1024 and fp16. The other experimental setups follow most of the training techniques used in DeiT [30], and additional details can be found in Appendix A. Note that our DiffRate is highlighted in the tables in gray, and **bold** denotes the best results.

6.2. Comparison with state-of-the-art

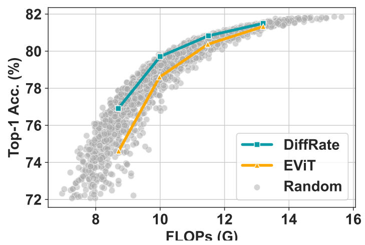
Compression Schedule. In Fig. 4, we compare the compression rate obtained by DiffRate with three other token pruning schedules, including EViT [41], ToMe [1], and randomly sampled schedules. We evaluate the FLOPs and accuracy on the ImageNet-1k validation dataset using an off-the-shelf ViT-B (DeiT) and investigate three token compression options: only pruning, only merging, and a combination of pruning and merging, as depicted in Sec. 4.1. We can

Table 1: **Token compression on the off-the-shelf models.** We directly apply EViT [21], ToMe [1], and the proposed DiffRate to off-the-shelf models, *i.e.* without updating the network parameters of pre-trained models.

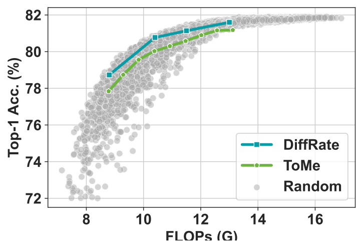
Model	Method	FLOPs	imgs/s	Acc.
ViT-S (DeiT)	Baseline [30]	4.6	5039	79.82
	EViT [21]	2.3	8950	73.83
	ToMe [1]	2.3	8874	77.99
	DiffRate	**2.3**	8901	**78.76**
	EViT [21]	3.0	6807	78.50
	ToMe [1]	2.9	6712	78.89
	DiffRate	**2.9**	6744	**79.58**
ViT-B (DeiT)	Baseline [30]	17.6	2130	81.83
	EViT [21]	8.7	4230	74.61
	ToMe [1]	8.8	4023	77.84
	DiffRate	**8.7**	4124	**78.98**
	EViT [21]	11.5	2886	80.37
	ToMe [1]	11.5	2834	80.58
	DiffRate	**11.5**	2865	**81.50**
ViT-B (MAE)	Baseline [13]	17.6	2130	83.72
	EViT [21]	8.7	4230	75.15
	ToMe [1]	8.8	4023	78.86
	DiffRate	**8.7**	4150	**79.96**
	EViT [21]	11.5	2886	82.01
	ToMe [1]	11.5	2834	82.32
	DiffRate	**11.5**	2865	**82.91**
ViT-L (MAE)	Baseline [13]	61.6	758	85.95
	EViT [21]	29.7	1672	81.52
	ToMe [1]	31.0	1550	84.24
	DiffRate	**31.0**	1580	**84.65**
	EViT [21]	39.6	1089	85.06
	ToMe [1]	42.3	1033	85.41
	DiffRate	**42.3**	1045	**85.56**
ViT-H (MAE)	Baseline [13]	167.4	299	86.88
	ToMe [1]	92.9	500	86.01
	EViT [21]	99.1	512	85.54
	DiffRate	**93.2**	504	**86.40**
	ToMe [1]	103.4	442	86.29
	EViT [21]	112.9	432	86.32
	DiffRate	**103.4**	450	**86.72**
LV-ViT-S	Baseline [16]	6.6	3630	83.30
	EViT [21]	3.9	5077	79.77
	DiffRate	**3.9**	5021	**82.56**

observe that DiffRate performed almost optimally, regardless of the token compression setting and FLOPs constraint. Additionally, DiffRate's advantage was more prominent at lower FLOPs constraints, indicating its ability to provide more appropriate compression rates at larger solution space. DiffRate also benefited from unified token compression, indicating that it can preserve more information by combining the merits of token merging and pruning.

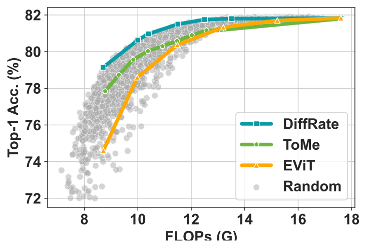
Off-the-shelf. We compare the proposed DiffRate with EViT [41] and ToMe [1] in the "off-the-shelf" setting, indicating direct use of the pre-trained model without updating the network parameters. Notably, the compression rate



(a) Pruning



(b) Merging



(c) Pruning and Merging

Figure 4: **Token Compression Schedule Comparison.** We test the proposed DiffRate on ViT-B (DeiT) with three token compression options: (a) Only token pruning like EViT [41], (b) Only token merging like ToMe [1],(c) Pruning and Merging as depicted in Sec. 4.1. The token compression schedules searched through DiffRate outperform the constant schedules used in ToMe [1] and EViT [21]. Moreover, the performance of our method is close to optimal when compared to $10,000$ randomly sampled schedules.

Table 2: **Token compression with training.** † indicates fine-tuning the model with searched compression rate for 30 epochs.

Model	Method	FLOPs	imgs/s	Acc.
ViT-S (DeiT)	Baseline [30]	4.6	5039	79.82
	DynamicViT [27]	2.9	6527	79.30
	Evo-ViT [36]	3.0	6679	79.40
	EViT [21]	3.0	6807	79.50
	ToMe [1]	2.9	6712	79.49
	ATS [8]	2.9	-	79.70
	SPViT [18]	2.6	-	79.34
	DiffRate	**2.9**	6744	**79.58**
	DiffRate†	**2.9**	6744	**79.83**
ViT-B (DeiT)	Baseline [30]	17.6	2130	81.83
	EViT [21]	11.5	2886	81.30
	ToMe [1]	11.5	2834	81.41
	DiffRate	**11.5**	2865	**81.50**
	DiffRate†	**11.5**	2865	**81.71**
ViT-B (MAE)	Baseline [13]	17.6	2130	83.72
	ToMe [1]	11.5	2834	82.94
	DiffRate	**11.5**	2865	**82.91**
	DiffRate†	**11.5**	2865	**83.25**
ViT-L (MAE)	Baseline [13]	61.6	758	85.95
	ToMe [1]	42.3	1033	85.59
	DiffRate	**42.3**	1045	**85.56**
	DiffRate†	**42.3**	1045	**85.71**
ViT-H (MAE)	Baseline [13]	167.4	299	86.88
	ToMe [1]	103.4	442	86.51
	DiffRate	**103.4**	450	**86.72**

achieved by DiffRate can be applied to pre-trained models without any additional computation. As shown in Table 1, DiffRate consistently outperforms both EViT [21] and ToMe [1] across various models. Specifically, DiffRate outperforms ToMe [1] by 0.14% to 1.14%, and outperforms EViT [21] by 0.39% to 4.93%. These results demonstrate the strong potential of DiffRate as an effective post-training token compression method. More results at other FLOPs constraints can be found in Appendix C.

With Training. In Table 2, we compare the proposed DiffRate with several methods that require training the model, indicating fine-tuning with pre-trained models or training from scratch. The evaluated methods include EViT [1] and ToMe [1], which train models from scratch, and ATS [8], DynamicViT [27], SP-ViT [18], which fine-tune pre-trained models for 30 epochs. To ensure a fair comparison, for DiffRate, we also fine-tune the pre-trained models for 30 epochs with the searched compression rate. We can observe that DiffRate still maintains a performance advantage compared to methods that require training. Specifically, ATS achieves a top-1 accuracy of 79.70% in DeiT-S, close to our 79.83%. However, it is important to note that ATS is an input-adaptive token pruning method that cannot be applied to batch inference, while DiffRate does not suffer this problem. Moreover, we find that DiffRate utilized with an off-the-shelf model achieves comparable or superior performance to methods that require training. For instance, DiffRate attains 79.58% on the off-the-shelf DeiT-S [30], while EViT and ToMe, after training, only achieve 79.50% and 79.49%, respectively. Similar results are also observed in ViT-B (DeiT) and ViT-B/L/H (MAE).

Table 3: **Results under multiple complexity constraints.** on ViT-S (DeiT). DiffRate indicates single FLOPs constraint as Eqn. (13), DiffRate-M indicates multiple complexity constraint as Eqn. (15).

Method	FLOPs(G)	Latency(ms)	Power(mW)	Acc.
Baseline	4.6	68.1	156	79.82
EViT	3.0	40.4	99	79.50
DiffRate	2.9	40.1	98	**79.83**
DiffRate-M	**2.9**	**37.6**	**90**	79.80

Multiple Complexity Constraints. In addition, we supervise DiffRate with three computational complexity constraints: FLOPs, latency, and power, as detailed in Eqn. (15). As shown in Table 3, integrating multiple complexity constraints enhances the trade-off between performance and complexity. Specifically, leveraging multiple complexity constraints leads to a remarkable reduction of

Table 4: **Ablation experiments** using ViT-B (DeiT) [30]. Our default settings are marked in gray.

Metric	Acc.(%)
Random	79.72
$\mathbf{A_i}$	81.38
$\mathbf{A_c} \cdot \mathbf{\lvert V \rvert}$	**81.53**
$\mathbf{A_c}$	81.50

(a) **Sorting Metric.** Simply class attention can measure the importance of tokens.

Option	Acc.(%)
Pruning	80.83
Merging	81.14
Merging-Pruning	81.18
Pruning-Merging	**81.50**

(b) **Token Compression Module Option.** A merging and pruning pipeline is the best choice.

Number	Acc.(%)
1,000	81.40
4,000	81.46
16,000	**81.50**
All	**81.50**

(c) **Training Data.** 1,000 images is enough to optimize compression rate.

Time (g-hrs)	Acc.(%)
0.9 (1-ep)	81.32
2.7 (3-ep)	81.50
9 (10-ep)	81.50
27 (30-ep)	**81.52**

(d) **Optimization Time.** Token compression rate can converge within 2.7 gpu houres.

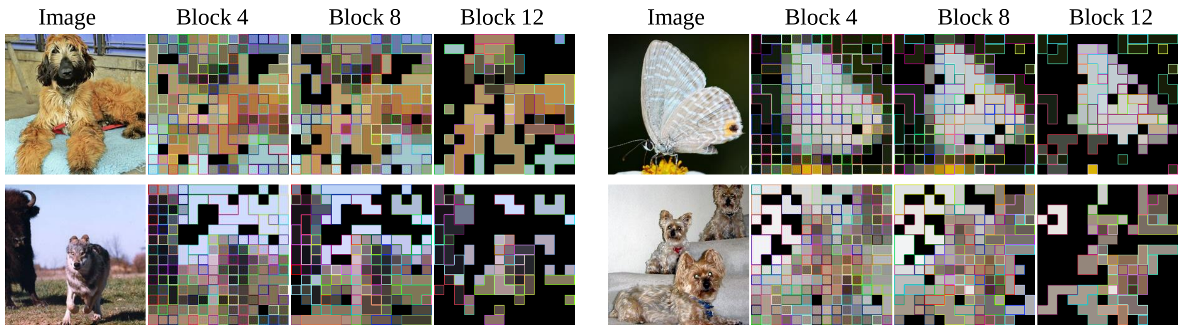


Figure 5: **Image Visualizations.** Results of proposed DiffRate on ImageNet-1k validation using a pre-trained DeiT-B model, only 34 tokens left in block 12. Merged tokens are represented by patches with the same inner and border color, while pruned tokens are represented by black. The visualizations show that DiffRate can gradually prune the redundant token in the background and merge less-discriminative tokens in the foreground.

2.5ms in latency and 8mW in power consumption compared to a single FLOPs constraint, with only a negligible decline in accuracy of a mere 0.03%.

6.3. Ablation Study

In this section, we perform DiffRate with variants to investigate the effectiveness of our proposed method. As the experiment shown in Table 4a, we compare the influence of several token sorting metrics, including randomly generated rank, class attention (A_c), image attention (A_i) [21], and the product of class attention and the value matrix's norm ($A_c \cdot |V|$) [8]. It can be observed that A_c and $A_c \cdot |V|$ exhibit similar performance, and we choose A_c as our default setting since it does not require any additional computation. Then Table 4b compares several token compression options, including only pruning, only merging, pruning then merging, and merging then pruning. The results show that pruning then merging performs the best since it successfully combines the advantages of both pruning and merging. What's more, in Table 4c, we investigate the amount of training data required by DiffRate to find the optimal compression rate. Surprisingly, we find that only $1,000$ images are sufficient to obtain an appropriate compression rate. Although we also optimize the token compression rate using the entire training dataset, our results demonstrate the potential for DiffRate to work well even with minimal data. Lastly, Table 4d investigates the convergence time required by DiffRate. We can find that only three epochs are required, demonstrating the efficiency of DiffRate as a token compression approach.

6.4. Visualization

The visualizations results in Fig. 5 demonstrate that DiffRate effectively removes semantically irrelevant background information. Furthermore, DiffRate can reduce the number of tokens by merging less-discriminative tokens in the foreground. For example, in the first row, DiffRate successfully removes most of the background and merges the dog hair and butterfly wings tokens into fewer tokens. In the second row, DiffRate preserves salient information tokens in different image regions, even when multiple instances exist. Overall, the visualization results highlight the effectiveness of our proposed DiffRate method in compressing ViT models without significant information loss. See more results in Appendix D.

7. Conclusion

This work presents a new token compression framework, named Differentiable Compression Rate (DiffRate). The proposed approach integrates both token pruning and merging into a unified framework that can optimize the compression rate in a differentiable manner. To achieve this, we introduced a novel Differentiable Discrete Proxy (DDP) module that can effectively determine the optimal compression rate using gradient back-propagation. Our experimental results demonstrate that DiffRate can perform comparable or superior to previous state-of-the-art token compression methods, even without fine-tuning the model. Additionally, DiffRate is highly data-efficient, as it can identify the appropriate compression rate using only $1,000$ images. Overall, the proposed DiffRate framework offers a new perspective on token compression by revealing the importance of compression rate. We believe that this approach has the potential to pave the way for further advancements in token compression research.

Acknowledgement

This work is supported by National Key R&D Program of China (No.2022ZD0118201) , the National Science Fund for Distinguished Young Scholars (No.62025603), the National Natural Science Foundation of China (No. U21B2037, No. U22B2051, No. 62176222, No. 62176223, No. 62176226, No. 62072386, No. 62072387, No. 62072389, No. 62002305 and No. 62272401), and the Natural Science Foundation of Fujian Province of China (No.2021J01002, No.2022J06001). This work is also partially supported by the National Key R&D Program of China(NO.2022ZD0160100), and in part by Shanghai Committee of Science and Technology (Grant No. 21DZ1100100).

References

[1] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your ViT but faster. In *International Conference on Learning Representations*, 2023. 1, 2, 3, 6, 7, 13

[2] Daniel Bolya and Judy Hoffman. Token merging for fast stable diffusion. *arXiv*, 2023. 16

[3] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018. 2, 3

[4] Arnav Chavan, Zhiqiang Shen, Zhuang Liu, Zechun Liu, Kwang-Ting Cheng, and Eric P Xing. Vision transformer slimming: Multi-dimension searching in continuous optimization space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4931–4941, 2022. 2

[5] Mengzhao Chen, Mingbao Lin, Ke Li, Yunhang Shen, Yongjian Wu, Fei Chao, and Rongrong Ji. Cf-vit: A general coarse-to-fine method for vision transformer. *arXiv preprint arXiv:2203.03821*, 2022. 4

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 6

[7] 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. In *International Conference on Learning Representations (ICLR)*, 2020. 1, 13

[8] Mohsen Fayyaz, Soroush Abbasi Kouhpayegani, Farnoush Rezaei Jafari, Eric Sommerlade, Hamid Reza Vaezi Joze, Hamed Pirsiavash, and Juergen Gall. Adaptive token sampling for efficient vision transformers. *European Conference on Computer Vision (ECCV)*, 2022. 2, 7, 8

[9] Christoph Feichtenhofer, Haoqi Fan, Yanghao Li, and Kaiming He. Masked autoencoders as spatiotemporal learners. *arXiv preprint arXiv:2205.09113*, 2022. 1

[10] Hasan Genc, Seah Kim, Alon Amid, Ameer Haj-Ali, Vighnesh Iyer, Pranav Prakash, Jerry Zhao, Daniel Grubb, Harrison Liew, Howard Mao, Albert Ou, Colin Schmidt, Samuel Steffl, John Wright, Ion Stoica, Jonathan Ragan-Kelley, Krste Asanovic, Borivoje Nikolic, and Yakun Sophia Shao. Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 769–774, 2021. 6, 12

[11] Shaopeng Guo, Yujie Wang, Quanquan Li, and Junjie Yan. Dmcp: Differentiable markov channel pruning for neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1539–1547, 2020. 2, 3

[12] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. A survey on vision transformer. *IEEE TPAMI*, 2022. 1

[13] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009, 2022. 1, 2, 6, 7, 12

[14] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012. 5

[15] Ding Jia, Kai Han, Yunhe Wang, Yehui Tang, Jianyuan Guo, Chao Zhang, and Dacheng Tao. Efficient vision transformers via fine-grained manifold distillation. *arXiv preprint arXiv:2107.01378*, 2021. 2

[16] Zi-Hang Jiang, Qibin Hou, Li Yuan, Daquan Zhou, Yujun Shi, Xiaojie Jin, Anran Wang, and Jiashi Feng. All tokens matter: Token labeling for training better vision transformers. *Advances in neural information processing systems*, 34:18590–18602, 2021. 6

[17] Minsoo Kang and Bohyung Han. Operation-aware soft channel pruning using differentiable masks. In *International Conference on Machine Learning*, pages 5122–5131. PMLR, 2020. 3

[18] Zhenglun Kong, Peiyan Dong, Xiaolong Ma, Xin Meng, Wei Niu, Mengshu Sun, Xuan Shen, Geng Yuan, Bin Ren, Hao Tang, et al. Spvit: Enabling faster vision transformers via latency-aware soft token pruning. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XI*, pages 620–640. Springer, 2022. 2, 3, 7

[19] Ling Li, David Thorsley, and Joseph Hassoun. Sait: Sparse vision transformers through adaptive token pruning. *arXiv preprint arXiv:2210.05832*, 2022. 2

[20] Yanyu Li, Geng Yuan, Yang Wen, Eric Hu, Georgios Evangelidis, Sergey Tulyakov, Yanzhi Wang, and Jian Ren. Efficientformer: Vision transformers at mobilenet speed. *arXiv preprint arXiv:2206.01191*, 2022. 13

[21] Youwei Liang, Chongjian Ge, Zhan Tong, Yibing Song, Jue Wang, and Pengtao Xie. Not all patches are what you need: Expediting vision transformers via token reorganizations. In *International Conference on Learning Representations (ICLR)*, 2022. 1, 2, 3, 4, 6, 7, 8

[22] Mingbao Lin, Mengzhao Chen, Yuxin Zhang, Ke Li, Yunhang Shen, Chunhua Shen, and Rongrong Ji. Super vision transformer. *arXiv preprint arXiv:2205.11397*, 2022. 2

[23] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 2, 3

[24] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, pages 10012–10022, 2021. 1, 13

[25] Zhenhua Liu, Yunhe Wang, Kai Han, Wei Zhang, Siwei Ma, and Wen Gao. Post-training quantization for vision transformer. *Advances in Neural Information Processing Systems*, 34:28092–28103, 2021. 2

[26] Dmitrii Marin, Jen-Hao Rick Chang, Anurag Ranjan, Anish Prabhu, Mohammad Rastegari, and Oncel Tuzel. Token pooling in vision transformers. *arXiv preprint arXiv:2110.03860*, 2021. 2

[27] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 2, 3, 4, 5, 7

[28] Michael S Ryoo, AJ Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. Tokenlearner: What can 8 learned tokens do for images and videos? *arXiv preprint arXiv:2106.11297*, 2021. 2

[29] Yehui Tang, Kai Han, Yunhe Wang, Chang Xu, Jianyuan Guo, Chao Xu, and Dacheng Tao. Patch slimming for efficient vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12165–12174, 2022. 2

[30] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning (ICML)*, pages 10347–10357, 2021. 1, 6, 7, 8, 11, 12

[31] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 568–578, 2021. 16

[32] Zhenyu Wang, Hao Luo, WANG Pichao, Feng Ding, Fan Wang, and Hao Li. Vtc-lfc: Vision transformer compression with low-frequency components. In *Advances in Neural Information Processing Systems*, 2022. 2

[33] Kan Wu, Jinnian Zhang, Houwen Peng, Mengchen Liu, Bin Xiao, Jianlong Fu, and Lu Yuan. Tinyvit: Fast pretraining distillation for small vision transformers. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXI*, pages 68–85. Springer, 2022. 2

[34] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. In *NeurIPS*, pages 12077–12090, 2021. 1

[35] Hu Xu, Juncheng Li, Alexei Baevski, Michael Auli, Wojciech Galuba, Florian Metze, Christoph Feichtenhofer, et al. Masked autoencoders that listen. *arXiv preprint arXiv:2207.06405*, 2022. 1

[36] Yifan Xu, Zhijie Zhang, Mengdan Zhang, Kekai Sheng, Ke Li, Weiming Dong, Liqing Zhang, Changsheng Xu, and Xing Sun. Evo-vit: Slow-fast token evolution for dynamic vision transformer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 2964–2972, 2022. 2, 7

[37] Huanrui Yang, Hongxu Yin, Pavlo Molchanov, Hai Li, and Jan Kautz. Nvit: Vision transformer compression and parameter redistribution. *arXiv preprint arXiv:2110.04869*, 2021. 2

[38] 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. 2

[39] Weihao Yu, Chenyang Si, Pan Zhou, Mi Luo, Yichen Zhou, Jiashi Feng, Shuicheng Yan, and Xinchao Wang. Metaformer baselines for vision. *arXiv preprint arXiv:2210.13452*, 2022. 13, 16

[40] Zhihang Yuan, Chenhao Xue, Yiqi Chen, Qiang Wu, and Guangyu Sun. Ptq4vit: Post-training quantization framework for vision transformers. *arXiv preprint arXiv:2111.12293*, 2021. 2

[41] Wang Zeng, Sheng Jin, Wentao Liu, Chen Qian, Ping Luo, Wanli Ouyang, and Xiaogang Wang. Not all tokens are equal: Human-centric visual analysis via token clustering transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11101–11111, 2022. 2, 6, 7, 16

[42] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *CVPR*, pages 633–641, 2017. 16

[43] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. In *ICLR*, 2022. 1

A. Training Details

Our training receipt follows most of the techniques used in DeiT [30]. In Table 5, we provide the default setting for compression rate searching. In this case, we initialize the backbone with their official pre-trained models, fix the network parameters, and train the proposed Differentiable Discrete Proxy for 3 epochs. In the first epoch, we optionally set λ_f in Eq. (13) to 0 for warm-up. To fine-tune the compressed models, we fix the compression rate and update the network parameters for 30 epochs. The default settings for fine-tuning are provided in Table 6. During compression rate searching, we employ attention masking to simulate token dropping, while in the fine-tuning process, we directly drop the redundant tokens. Any differences from the default DeiT recipe are highlighted in **bold** in the tables.

Table 5: Compression rate searching training settings.

cofig	value
optimizer	AdamW
learning rate	**0.01**
minimal learning rate	**0.001**
learning rate schedule	cosine decay
weight decay	**0**
batch size	1024
training epochs	**3**
augmentation	RandAug(9, 0.5)
LabelSmoth	0.1
DropPath	0.1
Mixup	0.8
CutMix	1.0

Table 6: Fine-tuning training settings.

cofig	value
optimizer	AdamW
learning rate	**2e-5**
minimal learning rate	**1e-6**
learning rate schedule	cosine decay
weight decay	0.05
batch size	1024
training epochs	**30**
augmentation	RandAug(9, 0.5)
LabelSmoth	0.1
DropPath	0.1
Mixup	0.8
CutMix	1.0

B. Computational Cost Constraint Details

This section provides more details about the computational cost constraint of DiffRate, including traditional FLOPs, and hardware-aware metrics like latency and power consumption.

B.1. FLOPs Calculation

Algorithm 2 FLOPs Calculation.

Input: block-wise pruning compression rate $\boldsymbol{\alpha}_p = \{\alpha_p^l\}_{l=1}^L$ and merging compression rate $\boldsymbol{\alpha}_m = \{\alpha_m^l\}_{l=1}^L$, embedding size C.

Output: FLOPs $\mathcal{F}(\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_m)$.

```
1: $\alpha^0 = 0$
2: $\mathcal{F}(\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_m) = 0$                      ▷ FLOPs
3: for l=1 to L do
4:     $\mathcal{F}(\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_m)$ += $4NC^2 + 2N^2C$          ▷ Attention
5:     $\alpha^l = max(\alpha^{l-1}, \alpha_p^l, \alpha_m^l)$
6:     $N = N(1-\alpha^l)$
7:     $\mathcal{F}(\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_m)$ += $8NC^2$                  ▷ MLP
8: end for
9: return $\mathcal{F}(\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_m)$
```

By utilizing the compression rate obtained from our proposed DiffRate, we can calculate the corresponding FLOPs $\mathcal{F}(\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_m)$ using Algorithm 2. The final FLOPs calculation also includes the patch embedding layer and classifier, which are excluded from Algorithm 2 for clarity. Additionally, we utilize the straight-through estimator (STE) for backpropagation in the max operation of Line 5.

Table 7: **Gemmini Search Space.** The Gemmini search space is determined by the number of tiles/meshes in each row and column, which indicates its computational resources, while the bank number/capacity of the scratchpad memory and accumulator determines its memory resources. The buswidth sets an upper limit on the communication speed between the scratchpad memory and computation modules.

parameters	type	search space
Tiles in a row	int	1,2,4,8
Tiles in a column	int	1,2,4,8
Meshes in a row	int	4,8,16,32
Meshes in a column	int	4,8,16,32
Buswidth (bit)	int	64,128,256,512
Bank number of scratchpad memory	int	1,2,4,8,16
Capacity of scratchpad memory (MB)	int	0.25,0.5,1,2,4
Capacity of accumulator (KB)	int	64,128,256,512,1024

B.2. Latency and Power Constraint

Apart from considering FLOPs, we can incorporate a differentiable method that accounts for hardware (HW) performance metrics, including latency and power consumption. Both latency and power consumption can be constrained in a similar manner. Therefore, we utilize the notation $\mathcal{L}_{hw}$

to represent both $\mathcal{L}_{la}$ and $\mathcal{L}_{pw}$ in Eq. (15) of the main text. The loss function of a hardware metric is given by:

$$\mathcal{L}_{hw} = \log(\cosh(E_{hw}(\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_m, \boldsymbol{\beta}) - T_{hw})) \tag{16}$$

where $\mathcal{L}_{hw}$ represents the corresponding HW performance constraint loss, E_{hw} is the HW performance with HW parameters $\boldsymbol{\beta}$ and compression rates $\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_m$, T_{hw} is the given target HW performance.

To find the optimal solution that considers both accuracy and HW performance metrics, we can co-explore the design space of compression ratio and HW simultaneously. We use Gemmini [10], a deep-learning accelerator generation framework that can produce a wide range of realistic accelerators from a flexible architecture template. The HW parameters $\boldsymbol{\beta}$ are provided in Table 7.

We adopt iterative training method to optimize compression rates and HW parameters. To be specific, we first optimize compression rate given HW parameters, and then optimize HW parameters given compression rates. To make the loss of hardware metric differentiable w.r.t. compression rates, we formulate E_{hw} as

$$E_{hw}^{\alpha} = \sum_{l=1}^{L} (\alpha^l + \mathrm{SG}(1-\alpha^l))\mathcal{F}'(\alpha^l, \boldsymbol{\beta}^*) \tag{17}$$

where $\alpha^l = max(\alpha^{l-1}, \alpha_p^l, \alpha_m^l)$, and $SG(\cdot)$ is the operator of stopping gradient. $\mathcal{F}'$ denotes the HW cost model, which can calculate a hardware metric given compression rate and HW parameters. '*' indicates that the HW parameters are fixed.

On the other hand, to make hardware metric differentiable w.r.t. compression rates, we formulate E_{hw} as

$$E_{hw}^{\beta} = \sum_{h=1}^{H} \sum_{l=1}^{L} (\beta_h + \mathrm{SG}(1-\beta_h))\mathcal{F}'(\alpha^{l*}, \boldsymbol{\beta}) \tag{18}$$

where $\beta_h = \mathrm{GS}(\pi_h)$, π_h is learnable parameter for h-th HW parameter, and GS represents Gumbel-Softmax function. Here '*' indicates that the compression rates are fixed.

With Eqn.(17) and Eqn.(18), we can update the compression rate and HW designs in a differentiable manner to satisfy the target latency and power consumption.

C. More Results

In this section, we present detailed results from our experiments. Firstly, we present the results obtained across different FLOPs constraints for the DeiT [30] in Sec. C.1 and for the MAE [13] in Sec. C.2. In Sec. C.3, we provide an detailed of the compression schedule that we searched. Additionally, we investigate the transferability of the searched compression schedules across different models in Sec. C.4. Finally, we demonstrate the effectiveness of the compressed models by training them from scratch with faster training speeds in Sec. C.5.

C.1. DeiT Models

Table 8 displays the comprehensive results of DeiT [30], encompassing both off-the-shelf models and fine-tuned models.

Table 8: **Full DeiT Results.** FT denotes fine-tuning the compressed model for 30 epochs. Gray denotes the official un-compressed pre-trained models (Baseline).

Model	FLOPs(G)	Acc.(%) w/o FT	Acc.(%) w/ FT
ViT-T (DeiT)	1.3	72.13	-
	0.6	70.36	71.11
	0.7	71.16	71.70
	0.8	71.74	72.18
	0.9	71.91	72.39
	1.0	72.12	72.46
ViT-S (DeiT)	4.6	79.82	-
	2.3	78.74	79.39
	2.5	79.09	79.61
	2.7	79.40	79.64
	2.9	79.58	79.83
	3.1	79.71	79.90
ViT-B (DeiT)	17.6	81.82	-
	8.7	78.98	80.61
	10.0	80.63	81.17
	10.4	80.97	81.30
	11.5	81.50	81.59
	12.5	81.75	81.80

C.2. MAE Models

Also, Table 9 presents the complete outcomes of MAE [13], encompassing both off-the-shelf models and fine-tuned models. We do not fine-tune ViT-H due to the constraints of computational resources.

C.3. Seached Compression Schedule

Table 10 presents the detailed compression schedule that we searched. Notably, in contrast to DeiT models, MAE models tend to retain more tokens in the deep block. This is because DeiT classifies solely based on the class token, while MAE classifies based on the average of all image tokens.

C.4. Compression Schedule Transfer

The number of blocks in ViT-T, ViT-S, and ViT-B is 12, facilitating the transferability of their block-wise compression rates. To investigate this, we transferred the compression rates obtained from ViT-T and ViT-B to ViT-S, as illustrated in Fig. 6. We can observe that the compression rate attained by ViT-S itself was optimal, whereas the transfer of compression rate from ViT-T to ViT-S resulted in

Table 9: **Full MAE Results.** FT denotes fine-tuning the compressed model for 30 epochs. Gray denotes the official un-compressed pre-trained models (Baseline).

Model	FLOPs(G)	Acc.(%)	
		w/o FT	w/ FT
ViT-B (MAE)	17.6	83.72	-
	8.7	79.96	81.89
	10.0	81.87	82.65
	10.4	82.07	82.83
	11.5	82.91	83.19
ViT-L (MAE)	61.6	85.95	-
	31.0	84.65	85.31
	34.7	85.19	85.45
	38.5	85.45	85.61
	42.3	85.56	85.63
	46.1	85.76	85.84
ViT-H (MAE)	167.4	86.88	-
	83.7	86.15	-
	93.2	86.40	-
	103.4	86.72	-
	124.5	86.77	-

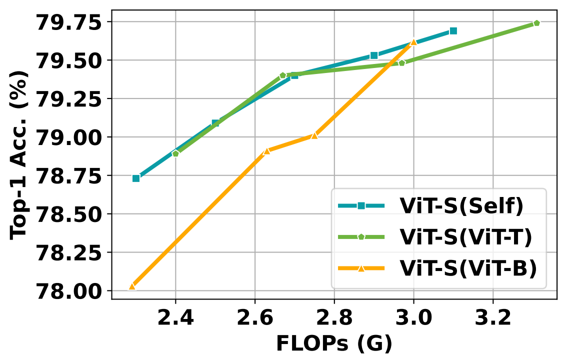


Figure 6: **Compression rate transfer**. Transferring the compression rate of ViT-B(DeiT) and ViT-T(DeiT) to ViT-S(DeiT). Self indicates the compression rate learn in ViT-S(DeiT).

a similar outcome. However, the transfer of compression rate from ViT-B to ViT-S leads to significantly poorer performance. This experiment verifies the ability of our proposed DiffRate to learn block-wise compression rates suitable for different network structures based on their features. Furthermore, it highlights that compression rates are somewhat transferable among similar network structures, such as transferring the compression rate from ViT-T to ViT-S.

C.5. Train from Scratch

The compressed model also has the capability to train from scratch using the searched compression rate. In this scenario, redundant tokens are directly eliminated, resulting in a faster training speed. As presented in Table 11, DiffRate yields a $1.4\times$ increase in training speed with only a -0.06% performance degradation.

D. More Visualization

We utilize the approach proposed by ToMe [1] to generate visualizations of the merging results. Specifically, we map each merged and pruned token back to its original input patch. To visualize merged tokens, we assign each input patch the average color of the merged tokens it belongs to and apply a random border color to distinguish tokens. For pruned tokens, we color their corresponding input patches black. Fig. 7 provides additional examples of token compression on images, extending the visualizations shown in Fig. 5. Our proposed DiffRate approach effectively identifies semantic objects, removes semantically irrelevant background tokens, and merges less-discriminative tokens in the foreground. By combining the advantages of pruning and merging through learnable compression rates, DiffRate can reduce the token count with minimal loss of information.

E. Overhead of DiffRate

In this section, we offer the analysis about computational overhead introduced by our proposed DiffRate framework. For a given number of input tokens, denoted by N, and a total of L blocks, the reparameterization trick introduces $2NL$ parameters and $\frac{(N^2+5N)L}{2}$ FLOPs. For example, we consider DeiT-S with $N = 196$ and $L = 12$, which results in only 4.7k additional parameters and 0.24M additional FLOPs. Furthermore, we evaluated the FLOPs of the DiffRate module within one block, taking into account the number of input tokens (N), the embedding size (C), and the number of merging tokens (N_m). The FLOPs of the DiffRate module within a block can be approximated as $N \log N + (N - N_m + 1)N_m C$. For instance, in the case of one DeiT-S block with $N = 196$, $C = 384$, and $N_m = 20$, the FLOPs of the DiffRate module amount to 1.36M. Overall, the overhead of DiffRate framework is negligible compared to the original 22M parameters and 4.6G FLOPs for DeiT-S.

F. Extending to Hierarchical Architecture

In addition to the standard ViT [7], various ViT variants have emerged, including EfficientFormer [20], Swin Transformer [24], CAFormer [39], among others. These modern ViT variants commonly employ a hierarchical architecture, dividing the network into multiple stages and progressively reducing the resolution of the feature map. Within hierarchical architectures, downsampling operations, such as convolution or pooling, are used to achieve the reduction in resolution. These operations rely on maintaining the spatial in-

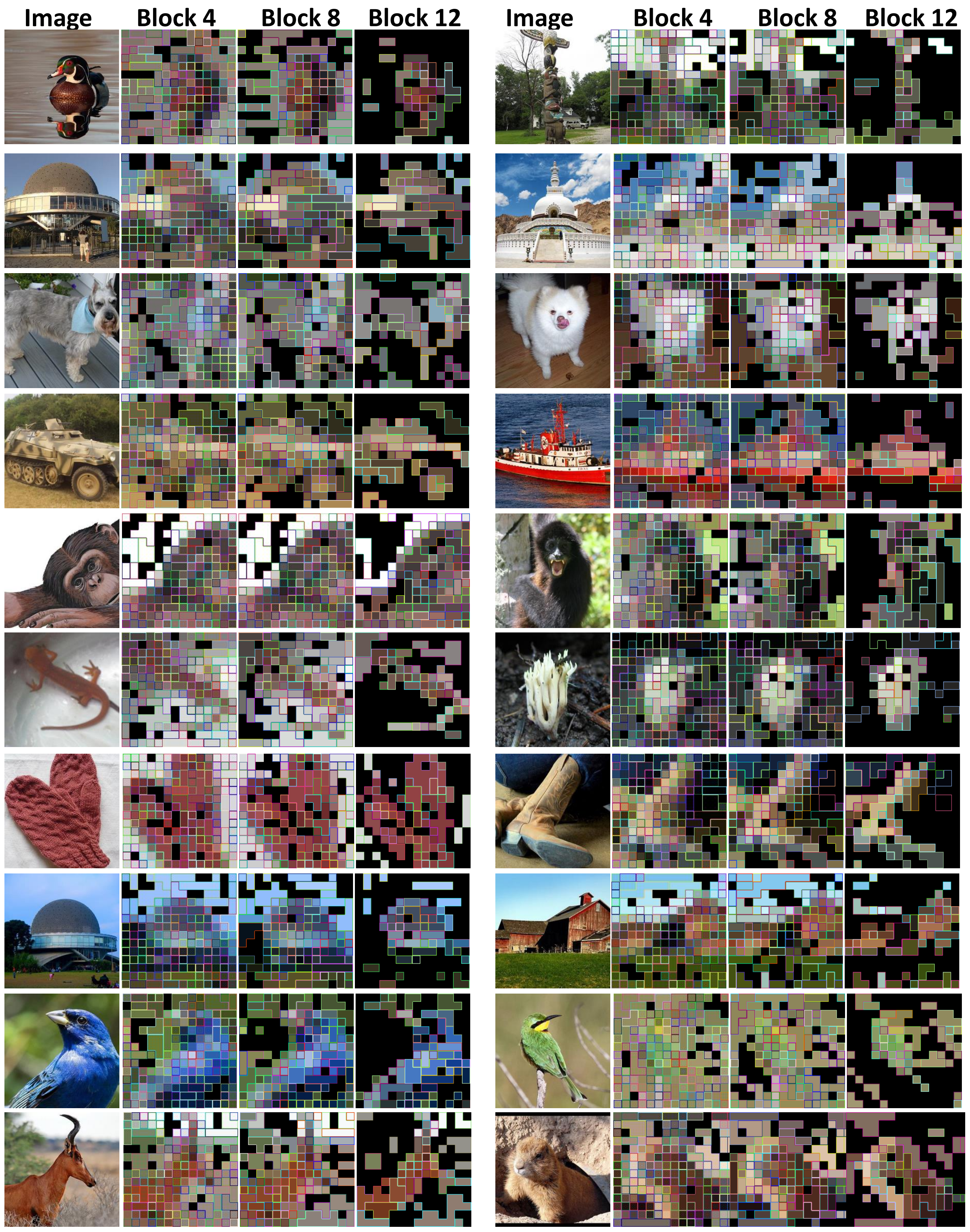


Figure 7: **More visualization.** Continuation of Fig 5.

Table 10: **Searched Compression Schedule.** We provide the block-wise kept tokens number for pruning and merging, respectively.

Model	FLOPs(G)	Compression Schedule Prune & Merge
ViT-T (DeiT)	0.6	[197,196,180,157,130,107,86,73,63,51,40,3]
		[197,187,167,139,114,90,76,66,57,45,37,3]
	0.7	[197,196,194,180,150,123,98,82,68,60,52,3]
		[197,196,192,158,133,103,88,72,64,58,49,3]
	0.8	[197,197,196,186,166,147,117,103,92,80,74,3]
		[197,196,190,172,154,125,107,96,84,78,70,3]
	0.9	[197,197,196,190,182,166,141,125,113,105,99,3]
		[197,196,194,188,172,147,129,115,107,103,96,3]
	1.0	[197,197,196,194,188,184,178,164,156,137,129,3]
		[197,197,196,190,186,154,148,156,145,135,125,3]
ViT-S (DeiT)	2.3	[197,192,168,143,121,105,92,74,62,45,33,3]
		[197,180,156,127,109,98,80,66,52,37,31,3]
	2.5	[197,192,174,156,131,115,111,99,76,50,39,3]
		[197,186,160,133,119,107,96,82,66,43,37,3]
	2.7	[197,196,182,160,141,127,115,105,88,66,49,3]
		[197,188,170,147,131,119,109,96,76,54,45,3]
	2.9	[197,196,190,168,150,139,129,117,99,78,58,3]
		[197,194,176,156,141,133,121,107,88,64,56,3]
	3.1	[197,197,194,180,160,147,137,129,115,92,76,3]
		[197,196,186,164,150,141,133,121,103,78,68,3]
ViT-B (DeiT)	8.7	[197,192,172,156,131,107,90,72,56,33,17,3]
		[197,178,162,141,115,96,78,60,47,21,13,3]
	10.0	[197,194,182,168,148,129,113,101,82,49,25,3]
		[197,186,174,156,137,117,105,90,66,31,19,3]
	10.4	[197,196,188,174,156,141,121,103,90,58,25,3]
		[197,190,184,164,145,131,107,94,74,31,21,3]
	11.5	[197,197,197,188,170,154,139,123,107,72,43,3]
		[197,197,192,178,158,145,127,111,94,50,35,3]
	12.5	[197,197,196,192,184,170,154,139,129,94,58,3]
		[197,196,194,190,176,160,145,131,115,68,47,3]
ViT-B (MAE)	8.7	[197,192,166,143,119,103,86,72,54,43,37,23]
		[197,176,150,127,107,92,76,58,47,41,23,23]
	10.0	[197,194,173,154,135,120,108,92,76,64,52,40]
		[197,181,160,141,122,116,95,80,67,60,40,40]
	10.4	[197,195,179,159,140,122,105,100,89,71,56,45]
		[197,187,166,145,125,109,102,97,75,64,45,45]
	11.5	[197,196,182,172,162,147,131,109,101,94,70,49]
		[197,192,178,164,150,133,115,101,96,82,49,49]

Table 11: **Train from scratch.** Train compressed ViT-S (DeiT) from scratch with the official DeiT training receipt.

Model	FLOPs	Throughput	Train Speed	Acc.
ViT-S (DeiT)	4.6	5039	1×	79.82
DiffRate	2.3	8901	1.8×	79.41
DiffRate	2.9	6744	1.4×	79.76

tegrity of the feature map. However, traditional token compression operations disrupt the spatial integrity by reducing the number of tokens in an unstructured manner. As a result, conventional token compression techniques cannot be directly applied to hierarchical architectures. To overcome this challenge, we propose a token uncompression module (see Fig.,8) that restores the compressed token sequence by copying tokens based on their relationships, inspired by the

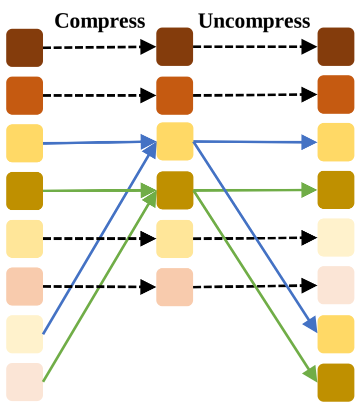


Figure 8: An example of token uncompression. We only consider token merging here.

Table 12: Appling DiffRate to CAFormer-S36 on ImageNet-1k.

Method	FLOPs(G)	Top-1 Acc.(%)	
		w/o tuning	w/ tuning
Baseline	8.0	84.45	-
DiffRate	6.0	84.21	84.32
	5.6	83.92	84.21
	5.2	83.49	84.03

Table 13: Downstream semantic segmentation task using Semantic FPN with CAFormer-S36 backbone on ADE20K dataset. FLOPs is calculated under the input scale of 512×704.

Backbone	FLOPs (G)	mIoU (%)	mAcc (%)
CAFormer-S36	77.2	41.05	51.50
w/ DiffRate	54.5	40.88	51.32

approaches in ToMeSD [2] and TCFormer [41]. By incorporating this module at the end of each stage, we can adapt our proposed DiffRate method to hierarchical architectures. Specifically, we apply DiffRate to the CAFormer [39], a state-of-the-art ViT variant with four stages. DiffRate is applied only to the third stage, as the first two stages consist of convolution blocks, and the last stage incurs minimal computational cost (6% in CAFormer-S36). As shown in Table 12, DiffRate achieves a significant 25% reduction in FLOPs with a marginal sacrifice of 0.24% in accuracy without additional fine-tuning. Moreover, when fine-tuned, DiffRate achieves a remarkable 35% reduction in FLOPs with a minimal accuracy drop of only 0.42%.

G. Extending to Downstream Tasks

By incorporating the introduced uncompression module in Fig. 8, DiffRate can also be applied to the model for downstream tasks. We begin by transferring the uncompressed pre-trained CAFormer-S36 model to ADE20K [42], following the settings of PVT [31]. Subsequently, DiffRate is applied to the third stage of the trained segmentation model without further fine-tuning. The results in Table,13 demonstrate that DiffRate achieves a significant 30% reduction in FLOPs with negligible performance degradation.