Spatio-Temporal Gated Transformers for Efficient Video Processing

Yawei Li[1]*, **Babak Ehteshami Bejnordi**[2], **Bert Moons**[2], **Tijmen Blankevoort**[2], **Amirhossein Habibian**[2], **Radu Timofte**[1], **Luc Van Gool**[1,3]
[1]ETH Zürich, [2]Qualcomm AI Research, [3]KU Leuven
{yawei.li, timofter, vangool}@vision.ee.ethz.ch
{behtesha, tijmen, ahabibia}@qti.qualcomm.com, bert.moons@axelera.ai

Abstract

We focus on the problem of efficient video stream processing with fully transformer-based architectures. Recent advances brought by transformers for image-based tasks inspires the research interests of applying transformers for videos. Yet, when applying image-based transformer solutions to videos, the computation becomes inefficient due to the redundant information in adjacent video frames. An analysis of the computation cost of the video object detection framework DETR identifies the linear layers as the major computation bottleneck. Thus, we propose dynamic gating layers to conduct conditional computation. With the generated binary or ternary gates, it is possible to avoid the computation for the stable background tokens in the video frames. The effectiveness of the dynamic gating mechanism for transformers is validated by experimental results. For video object detection, the FLOPs could be reduced by 48.3% without a significant drop of accuracy.

1 Introduction

Recently, transformers thrive as a new family of network architectures in computer vision. Recent transformers perform on par with or even better than convolutional neural networks (CNNs) for a couple of downstream vision tasks including image classification [9, 32, 22, 24], object detection [2, 41], semantic segmentation [36], and pose estimation [39]. Most of these new developments are targeted for image-based vision tasks. The promising performance of transformers for image tasks encourages the research interests of applying transformers for video stream processing [11, 29, 28].

When designing solutions for video stream processing, there are two research directions, *i.e.*improving accuracy or efficiency. The first direction is to utilize the temporal redundancy from adjacent frames to boost the performance of the network for single frame processing [3]. The second direction is to improve the efficiency of single-frame processing by avoiding redundant computation [12]. The focus of this paper is to design efficient transformer-based architectures for stream video processing.

When adapting single image-based transformers for videos, the computation for the stable background areas is still performed. This leads to redundant computation which can slow down the processing speed for single frames. To improve the processing efficiency, both spatial and temporal redundancy can be leveraged [35, 33, 12]. Yet, existing works are either designed for CNN architectures [35, 12] or utilize only one aspect of the redundancy [33]. In this paper, we propose to utilize both spatial and temporal redundancy to enable efficient transformer-based video object detection.

The second target of this paper is to design a fully transformer-based framework for video object detection. The prevailing object detection framework DETR [2] still uses ResNet [14] as the backbone.

*Work done as an intern at Qualcomm AI Research.

Machine Learning for Autonomous Driving Workshop at the 35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

On the other hand, the newly proposed transformers [32, 24] have achieved performances better than the classical ResNet. This inspires us to replace the CNN backbone in DETR with Swin transformer [24]. As such, we can focus on the analysis of the computation bottleneck of transformers and apply the same efficiency-improving technique to the whole network.

In this paper, we first analyze the computational cost of DETR with Swin transformer as the backbone network. This analysis reveals that the major computation burden lies in the linear layers. The matrix multiplication for the attention mechanism is computationally cheap. This directs the efficiency improving efforts to the linear layers. Thus, we upgrade the linear layers by exploiting the spatial and temporal redundancy in videos. Specifically, we design a gating layer for the linear layers in transformers. The gating layer takes the token features from the previous and current frames as input and generates binary gates for the tokens. The generated binary gates serve as an indicator of the background area and object area and are used to dynamically select whether to recompute the feature of a token or just to borrow the feature from the previous frame. For the $\mathbf{QKV}$ projection, we design a ternary gate by adding one zeroing state. In the zeroing state, the token is zeroed out, which could lead to a further reduction of computation in attention computation. To facilitate the back-propagation of gradients through the binary or ternary gating layers, the Gumbel-Softmax sampler is used [16, 27]. The generated binary or ternary gates are sparsified with L_1 loss. In addition, the computational complexity of the linear layer that the gating layer is applied to is used as an additional regularization factor to balance the compression ratio of different layers. Thus, the contributions of this paper are as follows.

- We propose a fully transformer-based architecture for video objection detection. The transformer network is adapted from an image-based transformer for efficient video stream processing by exploiting the spatial and temporal redundancy in videos.
- We propose binary and ternary gating layers which dynamically decide whether to recompute features for the tokens in the current frame or to select features from the previous frame.
- We use L_1 loss regularized by the computational complexity of a layer to sparsify the gates.
- Experimental results show that the proposed method could lead to a significant reduction of computation for video object detection with negligible or even without drop of accuracy.

2 Related Works

Transformers are designed for modeling long-range dependencies in language sequences. The computational complexity of the attention mechanism, in that case, increases quadratically with the number of tokens [34, 8]. Thus, the computation bottleneck is the attention mechanism. The core of the problem is, therefore, how to improve the efficiency of attention. Previous methods rely on local attention [18], sparsity [4] and low rank approximations [17] to simplify space and time complexity. Instead of relying on those priors, performers use fast attention via positive orthogonal random features to approximate Softmax attention kernels, which leads to a linear space and time complexity [5].

Due to their successful application in language modeling, transformers have been applied to vision tasks recently [2, 41, 9, 32, 37, 22, 24, 36, 31, 1, 10]. In this case, the length of sequences generated from images is significantly reduced, which reduces the cost of attention computation in the self-attention mechanism. As a result, linear projection and the feed forward network (FFN) become the computation bottleneck. Because of this, the efficient attention methods listed above will lead to less relative efficiency gains.

Previous methods to optimize efficiency are designed for images instead of temporal streams. For video-based applications, there are ample opportunities to dynamically reduce the cost of inference. More specifically, the cost can be reduced in any neural network by exploiting both temporal and spatial redundancies of video streams. Prior work typically either focuses on spatial [30] or temporal redundancies [12], but does not offer an elegant unified approach. In this paper, a novel unified approach to dynamic data-dependent gating of both spatial and temporal features is introduced for transformers.

Apart from the techniques discussed above, there are also other methods for efficient computing such as network pruning [13, 15, 26, 25, 19, 21], quantization [6, 7], and low-rank approximation [40, 20]. These approaches are orthogonal to this work.

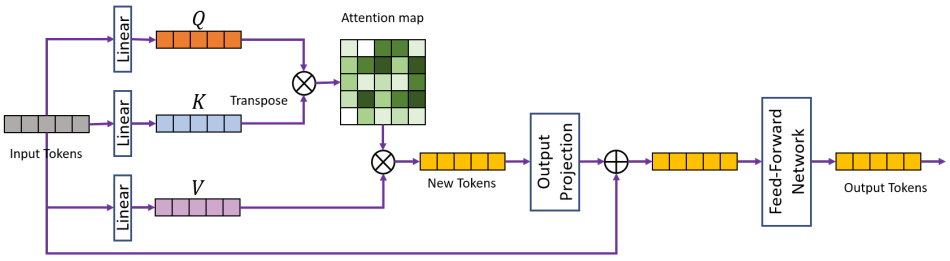

Figure 1: The transformer encoder architecture. It consists of a self-attention module and an FFN. Linear layers are used for the **QKV** projections, the output projection, and the FFN.

3 Preliminaries

In this section, the classic vision transformer for image processing is firstly described. Then the computation bottleneck in the self-attention module is analyzed. Based on that, an improved fully transformer-based framework for video object detection is described. At last, a breakdown analysis of the computational complexity of the new framework is carried out. This reveals the computationally heavy part of the framework and inspires the direction for efficient video stream processing.

3.1 Transformer encoder

Transformers are usually composed of a stack of encoders and decoders which have very similar architectures. For the simplicity of discussion, we mainly talk about the vanilla ViT encoders in this paper [9]. The discussion and conclusion could be generalized to decoders and other transformers (Swin transformer) easily.

Tokens as input. A transformer encoder consists of a self-attention module that models dependencies between tokens and an FFN that deepens the feature representation. Transformers take as input a sequence of tokens $\mathcal{X} = \{\mathbf{x}_i \in \mathbb{R}^d | i = 1, \ldots, N\}$, where N is the number of tokens and d is the embedding dimension. The sequence of tokens could be assembled into a matrix $\mathbf{X} \in \mathbb{R}^{N \times d}$. For image-based vision tasks, the input sequence is generated by converting the input image $\mathcal{I} \in \mathbb{R}^{C \times H \times W}$, where C, H, and W are the channel, height, and width of the input image, respectively.

Self-attention. The self-attention module models the relationship between the tokens by considering the similarities between them. The input tokens are first converted to a triplet of queries $\mathbf{Q}$, keys $\mathbf{K}$, and values $\mathbf{V}$ by linear projection layers, *i.e.*,

$$\mathbf{Y} = \mathbf{X}\mathbf{W}_Y + \mathbf{B}_Y, \tag{1}$$

where $\mathbf{Y} \in \{\mathbf{Q}, \mathbf{K}, \mathbf{V}\}$, $\mathbf{W}_Y \in \mathbb{R}^{d \times d}$, $\mathbf{B}_Y \in \mathbb{R}^d$. Then an attention map is derived by computing the similarities between the queries and keys and normalizing with Softmax function. The new token is computed as the weighted sum of the values with respect to the attention map, *i.e.*

$$\mathbf{X}_a = \mathrm{Softmax}(\mathbf{Q}\mathbf{K}^T/\sqrt{d})\mathbf{V}, \tag{2}$$

where the $\mathrm{Softmax}$ function is applied to the rows of the similarity matrix and d provides a normalization. An additional output projection layer is applied to the new tokens, *i.e.*

$$\mathbf{X}_o = \mathbf{X}_a\mathbf{W}_o + \mathbf{B}_o. \tag{3}$$

Only one head is used in the description above. In practice, more than one head could be used and features from the multiple heads could be concatenated along the embedding dimension. This constitutes the so-called Multiheaded Self-Attention (MSA) module.

Feed-forward network. The self-attention module is followed by an MLP with two fully-connected layers. The MLP deepens the feature representation and widens the hidden embedding dimension between the two fully-connected layers. That is,

$$\mathbf{X}_{\text{ffn}} = f((\mathbf{X}_o\mathbf{W}_1 + \mathbf{B}_1)\mathbf{W}_2) + \mathbf{B}_2, \tag{4}$$

where $\mathbf{W}_1 \in \mathbb{R}^{d \times \gamma d}$, $\mathbf{W}_2 \in \mathbb{R}^{\gamma d \times d}$, $\mathbf{B}_1 \in \mathbb{R}^{\gamma d}$, $\mathbf{B}_2 \in \mathbb{R}^d$, and $f(\cdot)$ denotes an activation function.

3.2 Computational complexity in self-attention

In the self-attention module, the computation is distributed to the linear projection layers in Eqn. 1 and Eqn. 3 and the matrix multiplication in Eqn. 2. The computational complexity of the linear projection layers and the matrix multiplication are $3Nd^2 + Nd^2 = 4Nd^2$ and $2N^2d$, respectively. Thus, the ratio of the computational complexity of the matrix multiplication in Eqn. 2 is $N/(2d + N)$. When N is large, the attention computation becomes the computation bottleneck. Thus, recent research focuses on improving the computational efficiency of the attention mechanism in Eqn. 2 [36, 24]. In this paper, we use Swin Transformer [24] as the backbone for vision tasks due to its high efficiency for attention computation in Eqn. 2. ViT [9] is also used as the backbone as a comparison.

3.3 Object detection with Swin transformer.

We showcase the adaptation of image-based transformers to efficient video stream processing with object detection as an example. The techniques and conclusions could be easily extended to other vision tasks such as recognition and semantic segmentation. We build our solution based on DETR [2] which is a recent end-to-end transformer-based solution for object detection.

Module		Params (%)	FLOPs (%)
Swin Backbone	Linear	19.10	29.14
	Attention	0.00	2.82
	FFN	38.16	52.16
Tail: Encoder	Linear	3.48	1.47
	Attention	0.00	2.48
	FFN	13.93	5.92
Tail: Decoder	Linear	6.95	1.00
	Attention	0.00	0.32
	FFN	13.93	0.69

Table 1: Breakdown analysis of the computational complexity of DETR. The resolution of the input image is 804×1054. "Linear" denotes the linear layers for $\mathbf{QKV}$ projection and output projection. This shows that the matrix multiplication in Eqn. 2 is quite cheap while most of the computation is consumed by the linear layers.

DETR is is not fully transformer-based network because of using ResNet [14] as the backbone. Meanwhile, recent research reveals that transformer architectures could easily outperform ResNet when used as the backbone for a couple of vision tasks including image recognition [32, 22], object detection [24], and semantic segmentation [36]. Thus, in this paper, we replace the backbone of DETR with the highly efficient Swin transformer. As such, the techniques developed for transformers in this paper could be applied to both the backbone transformer and the detector of DETR. This makes it possible to have a major efficiency improvement with a single technique.

3.4 Computational complexity of DETR

To identify the component where efficiency improvement efforts could be directed to, a breakdown analysis of the computational complexity of DETR is done in Table 1. As discussed in the previous subsection, Swin transformer is used as the backbone network. The detection tail is a transformer that contains 6 encoders and 6 decoders. We classify the model complexity of the submodules (backbone, encoder, decoder) into three groups including linear layers, attention computation, and MLP. As shown in Table 1, the attention computation is no longer the computationally heavy part of the framework since it only occupies 5.62% (2.82% + 2.48% + 0.32%) of the total computation. Most (90.38%) of the computation is consumed by the linear operations in the self-attention module and the FFN. This means that we should focus on the linear layers in transformers to improve the efficiency of video stream processing.

4 Dynamic Gating Mechanism

In this section, we propose to improve the processing efficiency of video streams by exploiting their spatial and temporal redundancy. The analysis in the previous section identifies that the computation bottleneck of DETR is the linear operations in the network. Thus, the efficiency improvement efforts are directed to the linear layers. Dynamic gating layers are introduced to improve the processing efficiency of the linear layers. The overview of the proposed dynamic gating layers is shown in Fig. 2. The dynamic gating layers are applied to the linear layers in the transformer network including the $\mathbf{QKV}$ projection layers, the output projection layer in the self-attention module, and the FFN.

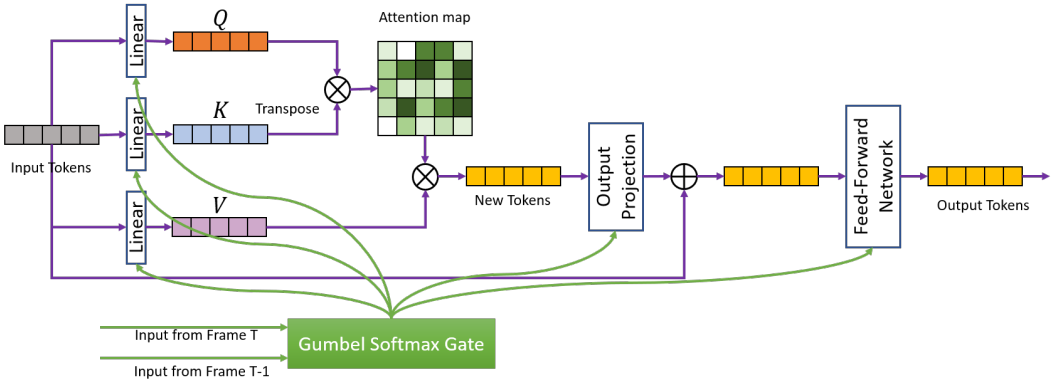

Figure 2: The position where the gating layers are attached. The dynamic gating layers are applied to the **QKV** projection layers, the output projection layer, and the FFN.

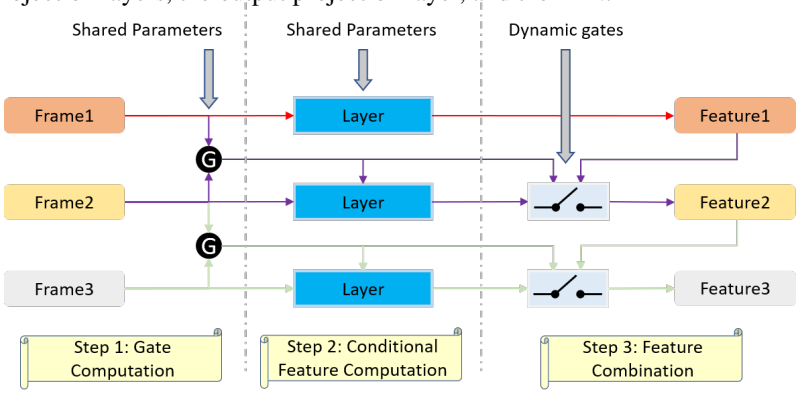

Figure 3: The video frame processing pipeline with the dynamic gating layers. The blue box denotes the linear layer where the gating layer is attached to.

The dynamic gating layer takes as input the features of two consecutive frames and emits binary or ternary gates as output. The generated gates indicate background and foreground moving object in the video frames. This information is used to avoid the redundant computation for the tokens in the background area which is assumed to be relatively static. To maintain the feature map, the features for the background tokens are borrowed from the previous frame.

4.1 Binary gates

The dynamic gating mechanism for video stream processing in a specific linear layer is shown in Fig. 3. The dynamic gating mechanism is applied to a linear layer denoted by the blue box in the figure. For the first frame, no gating mechanism is applied and the computation is done for every token no matter where (background or foreground) the token belongs to. Then starting from the second frame, the dynamic gating mechanism is applied. The gating layer takes the features from the current frame and the previous frame as input. It combines the information from the two frames and distinguishes the background area and foreground object (stationary and non-stationary regions), which is summarized in the output of the gating layer. If a token belongs to the foreground object (gate is on), then its feature is recomputed based on the information in the current frame. Otherwise, the feature for the token is just borrowed from the previous frame. In this way, the computation of the feature in the gated linear layers is done autoregressively, which can leads to a significant reduction of computation.

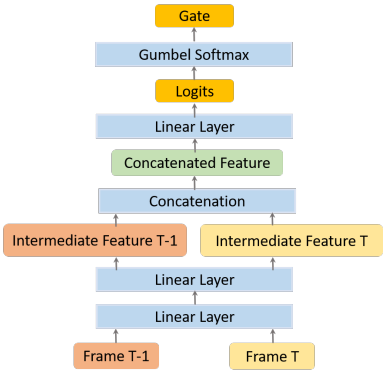

Figure 4: The architecture of the gating layers.

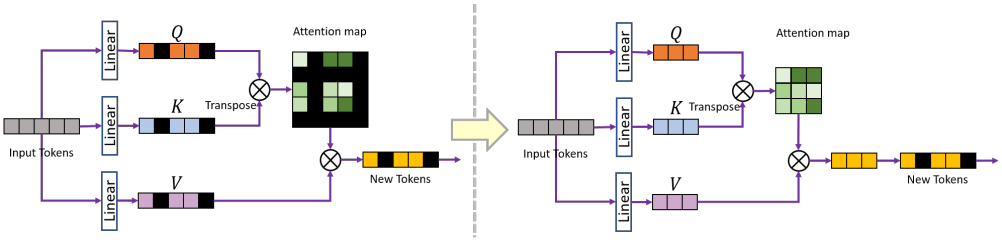

Figure 5: Benefits of zeroing mode in ternary gates for $\mathbf{QKV}$ projection layers.

The architecture of the gating layers is shown in Fig. 4. Two shared linear layers are first applied to extract features from the two input frames. Then the intermediate features are concatenated, which is input to another linear layer that fuses the information from two frames. To facilitate the training of the gating layer, the Gumbel Softmax sampler is applied to the output logits of the final layer. For the binary gate, the output logits have only one state for each token and can be denoted by a vector, *i.e.*

$$\mathbf{S} = (\mathbf{S}_1, \cdots, \mathbf{S}_i, \cdots, \mathbf{S}_N)^T, \tag{5}$$

where $\mathbf{S} \in \mathbb{R}^N$, i indexes the tokens. Then Sigmoid is applied to the logits, *i.e.*, $\mathbf{Z} = \mathrm{Sigmoid}(\mathbf{S}/T)$, where T is the temperature and is set to 2/3. The binary gate is derived by thresholding the gate state $\mathbf{Z}$ with respect to 0.5 in the forward path (and during inference), *i.e.*,

$$\mathbf{G}_i = \begin{cases} 1, & \mathbf{Z}_i > 0.5 \\ 0, & \mathbf{Z}_i \leq 0.5 \end{cases}. \tag{6}$$

The output feature is computed by selecting the feature from the current or the previous frame, *i.e.*

$$\mathbf{X}_g = \mathbf{X}_p \circ (1 - \mathbf{G}) + \mathbf{X}_c \circ \mathbf{G}, \tag{7}$$

where $\mathbf{X}_p$, $\mathbf{X}_c$, $\mathbf{X}_g$ denotes the feature from the previous frame, the feature from the current frame, and the gated feature, $\circ$ denotes Hadamard product. Note that to achieve an actual reduction of computation, the computation of $\mathbf{X}_c$ for $\mathbf{G} = 0$ could be skipped.

4.2 Ternary gates

We also design a ternary gate for the $\mathbf{QKV}$ projection layers in the self-attention module. The basic processing pipeline and architecture of the gating layers are the same as those of the binary gates. The only difference is that there are three states in the output logits of the gating layer, *i.e.*

$$\mathbf{S} = [\mathbf{S}_{:,1}, \mathbf{S}_{:,2}, \mathbf{S}_{:,3}], \tag{8}$$

where $\mathbf{S} \in \mathbb{R}^{N \times 3}$, $\mathbf{S}_{:,i}$ denotes the columns of $\mathbf{S}$. For each token i, the three states $\mathbf{S}_{i,1}$, $\mathbf{S}_{i,2}$, $\mathbf{S}_{i,3}$ correspond to zeroing, sharing, and computing mode, respectively. The sharing and computing modes are the same as those in binary gates. In zeroing mode, a token is just replaced by zeros. As shown in Fig. 5, a token in the zeroing state could just be removed. As a result, the attention computation in Eqn. 2 could be done on a smaller feature, which leads to a further reduction of computation. Yet, zeroing out tokens also leads to a loss of information. Thus, a trade-off between accuracy and efficiency should be made. A Softmax function is applied to the output logits, *i.e.*$\mathbf{Z} = \mathrm{Softmax}(\mathbf{S}/T)$. Then the ternary gate is determined by converting the state vector to a one-hot vector, *i.e.*

$$\mathbf{G}_{i,j} = \begin{cases} 1, & \mathbf{G}_{i,j} = \max_j \mathbf{G}_{i,j} \\ 0, & \text{otherwise} \end{cases}. \tag{9}$$

The final output feature is computed as

$$\mathbf{X}_g = \mathbf{X}_p \circ \mathbf{G}_{:,1} + \mathbf{X}_c \circ \mathbf{G}_{:,2}. \tag{10}$$

4.3 Loss function

To sparsify the binary and ternary gates, L_1 loss function is used. In addition, the loss term is regularized by the computational complexity of the linear layers. This could lead to a balanced compression of different layers. Thus, the loss function for the binary gate is calculated as

$$\mathcal{L}_b = \frac{1}{L} \sum_{l=1}^{L} \frac{\mathrm{FLOP}_l}{10^9} \gamma \mathrm{mean}(\mathbf{Z}^l), \tag{11}$$

Table 2: Experimental results for ternary gates. "Baseline" denotes the network without the proposed gating layers. Note that ternary gating layers are only applied to the $\mathbf{QKV}$ projection layers while binary gating layers are applied to the other linear layers. γ_1, γ_2, and γ_3 denotes the regularization factor for zeroing, sharing, and computing modes. For binary gating layers, γ_3 is used directly.

Backbone	γ_1	γ_2	γ_3	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	FLOPs	Params
Swin-T	Baseline			55.9	83.1	64.9	8.6	51.5	75.6	55.3G	45.2M
	0	0	5	55.9	82.9	64.9	8.4	51.6	75.6	32.6G	45.5M
	1	0	5	56.0	83.0	65.1	8.7	51.7	75.6	32.9G	45.5M
	5	0	5	56.0	83.1	65.1	8.6	51.7	75.8	33.9G	45.5M
	5	0	10	55.6	83.0	64.7	8.3	51.4	75.1	30.3G	45.5M
	5	0	15	55.3	82.9	64.2	8.2	51.2	74.3	28.6G	45.5M
ViT-Base	Baseline			43.5	75.7	45.2	8.7	38.1	66.5	52.3G	105.3M
	0	0	1	39.9	73.4	39.7	8.3	34.8	61.9	36.8G	105.7M
	0	0	3	39.6	73.3	39.2	8.1	34.9	61.0	30.0G	105.7M
	1	0	3	39.4	73.0	38.9	8.2	34.7	60.6	30.6G	105.7M
	1	0	5	39.6	73.4	38.9	8.1	34.9	60.6	27.2G	105.7M

where l is an index to the layers, γ is the regularization factor, FLOP_l denotes the computational complexity of a specific linear layer. The loss function for the ternary gate is computed as

$$\mathcal{L}_t = \frac{1}{L}\sum_{l=1}^{L}\frac{\text{FLOP}_l}{10^9}\big(\gamma_1\text{mean}(\mathbf{Z}^l_{:,1}) + \gamma_2\text{mean}(\mathbf{Z}^l_{:,2}) + \gamma_3\text{mean}(\mathbf{Z}^l_{:,3})\big), \tag{12}$$

where γ_1, γ_2, γ_3 are the regularization factors for the zeroing, sharing, and computing states.

5 Experimental Results

Dataset. The experimental results are shown in this section. The experiments are done on UA-DETRAC dataset [38], which is a challenging real-world video object detection dataset. The videos are recorded at frame rate of 25 and with resolution of 960×540. It consists of more than 140,000 frames and manually annotated 8250 vehicles. The average precision (AP) is used to evaluate the performance. Several versions are reported including the average over multiple IoU thresholds varying from 0.5 to 0.95 with a step size of 0.05, the AP with IoU threshold 0.5 and 0.75, and the AP for small, medium, and large objects. The number of parameters and FLOPs are also reported. FLOPs are averaged on the test subset.

Implementation details. We use DETR [2] as the framework for video object detection. The tiny version of Swin transformer (Swin-T) is used to replace the ResNet backbone of DETR. In addition, we also use ViT-Base as the backbone. ViT uses a fixed trainable positional encoding that is tailored to the ImageNet resolution 224×224. To use the pretrained weights for images with different resolution, the positional encoding is interpolated via bicubic interpolation. In addition, the class token for image classification is removed. The first linear layer in the gating layer reduces the number of features to 16. And this dimension is maintained until the last linear layer. This design ensures that the gating layer is small and does not bring too much parameter and FLOPs overhead. By default, the video streams are clipped to snippets with 8 frames. The first frame undergoes normal computation while starting from the second frame, gated computation is used. We also report using different number of frames in Table 4.

The network is first pre-trained on the MSCOCO dataset [23]. The setup of hyper-parameters is the same as DETR [2]. Then the network is fine-tuned on UA-DETRAC dataset. The learning rates of the backbone and the detection tail are set to 10^{-6} and 10^{-5}, respectively. The network is trained for 5 epochs and the learning rate is reduced by a factor of 10 at Epoch 4. After the training of 5 epochs, the dynamic gating layers are attached to the linear layers. Then the network is trained for another 5 epochs. The backbone and the detection tail are frozen during this training. The learning rate for the gating layers is 10^{-3} and is decayed at Epoch 4 by a factor of 10.

Ternary gates. The result of the ternary gating layers is shown in Table 2. As mentioned in SubSec. 4.2, the ternary gates are only applied to $\mathbf{QKV}$ projection layers. For the output projection layer in the attention module and the FFN, binary gating layers are applied instead. Different combinations of the regularization factors γ_1, γ_2, γ_3 are reported. Note that, we tried to penalize

Table 3: Experimental results for binary gates. "Baseline" denotes the network without the proposed gating layers. Binary gates are applied to all of the linear layers.

Backbone	γ	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	FLOPs	Params
Swin-T	Baseline	55.9	83.1	64.9	8.6	51.5	75.6	55.3G	45.2M
	0	56.0	83.2	65.0	8.9	51.6	75.9	56.6G	45.5M
	5	56.0	83.2	65.0	9.0	51.6	75.7	33.4G	45.5M
	10	55.7	83.1	64.7	9.0	51.4	75.4	30.7G	45.5M
	15	55.4	83.1	64.3	9.0	51.2	74.7	29.0G	45.5M
	30	54.6	83.0	63.2	8.8	50.6	73.2	26.4G	45.5M
	50	53.5	82.9	61.3	8.9	49.8	71.2	24.6G	45.5M
	100	51.4	82.5	57.3	9.2	48.1	67.7	22.2G	45.5M
ViT-Base	Baseline	43.5	75.7	45.2	8.7	38.1	66.5	52.3G	105.3M
	0	43.6	75.7	45.5	8.7	38.3	66.6	50.4G	105.7M
	5	43.0	75.6	44.5	8.7	37.8	65.4	28.4G	105.7M
	10	42.3	75.4	43.4	8.6	37.3	64.2	25.7G	105.7M
	15	41.9	75.3	42.6	8.6	36.9	63.4	24.4G	105.7M
	30	40.8	74.9	40.5	8.6	36.1	61.4	22.0G	105.7M
	50	39.6	74.4	38.1	8.4	35.1	59.5	20.3G	105.7M

the computing mode as well as the zeroing mode. This is because zeroing out too many tokens will deteriorate the performance of the network.

When Swin transformer is used as the backbone network, the additional number of parameter brought by the gating layers is only 0.3M, which only accounts for 0.66% of the original parameter count. The performance of the gated network remains almost the same as the baseline network but with significant reduction of computation. For example, when the computation is reduced by 38.7%, AP is even improved by 0.1% while AP_{50} and AP_L are increased by 0.2%. When the computation is reduced by 48.3%, there is only a negligible accuracy drop of 0.6%.

ViT-Base baseline consumes slightly fewer computation (3.0G) than the Swin-T baseline while has much more parameters. The AP of the ViT-Base baseline is 43.5%, which is much lower than the Swin-T baseline. This is consistent with the previous results because Swin transformer is much more efficient than ViT [24]. The performance of the gated network does not perform quite well when ViT-Base is used as the backbone network. In all of the reported hyperparameter setups, the accuracy of the gated networks is reduced by more than 3%. This shows that the $\mathbf{QKV}$ projection layers of ViT are more sensitive to the zeroing-out of tokens compared with Swin transformer.

Binary gates. In Table 3, we show the results when binary gates are applied to all of the linear layers. Different regularization factors γ are reported. For Swin transformer, binary gates behaves almost the same as ternary gates. When no gate loss is applied, the computation of the network is increased slightly from 55.3G to 56.6G. And this increase is brought by the gating layers. For ViT-Base, the binary gates perform much better than ternary gates. For example, when $\gamma = 5$, the computation is reduced by 45.7% while the accuracy is only reduced by 0.6%. Different regularization factors leads to a trade-off between network accuracy and efficiency. For both Swin transoformer and ViT, when the computation is reduced by nearly 50% ($\gamma = 15, 30$ for Swin transformer, $\gamma = 10$ for ViT), the accuracy drop is relatively small. In Table 4, we also report the results for different number of frames. When more frames are used, there is slightly more reduction of computation.

6 Conclusion

In this paper, we propose to adapt transformer networks designed for image-based vision tasks for efficient video stream processing. This adaption is based on the spatial and temporal redundancy in video streams. Specifically, video object detection is used as the example case. The developed techniques could be applied to other tasks such as video semantic segmentation. Binary and ternary gating layers are designed for the linear layers in the transformer-based object detection networks. The dynamic gating layers could avoid redundant computation in the background area in video frames. Experimental result shows that the computation of the video object detection network could be reduced significantly without any significant accuracy drop.

References

[1] J. Cao, Y. Li, K. Zhang, and L. Van Gool. Video super-resolution transformer. *arXiv preprint arXiv:2106.06847*, 2021.

[2] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. In *Proc. ECCV*, pages 213–229. Springer, 2020.

[3] Y. Chen, Y. Cao, H. Hu, and L. Wang. Memory enhanced global-local aggregation for video object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10337–10346, 2020.

[4] R. Child, S. Gray, A. Radford, and I. Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[5] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[6] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Proc. NeurIPS*, pages 3123–3131, 2015.

[7] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[10] P. Esser, R. Rombach, and B. Ommer. Taming transformers for high-resolution image synthesis. *arXiv preprint arXiv:2012.09841*, 2020.

[11] R. Girdhar, J. Carreira, C. Doersch, and A. Zisserman. Video action transformer network. In *Proc. CVPR*, pages 244–253, 2019.

[12] A. Habibian, D. Abati, T. S. Cohen, and B. E. Bejnordi. Skip-convolutions for efficient video processing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2695–2704, 2021.

[13] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *Proc. ICLR*, 2015.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. CVPR*, pages 770–778, 2016.

[15] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *Proc. ICCV*, pages 1389–1397, 2017.

[16] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[17] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

[18] N. Kitaev, Ł. Kaiser, and A. Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[19] Y. Li, S. Gu, C. Mayer, L. Van Gool, and R. Timofte. Group sparsity: The hinge between filter pruning and decomposition for network compression. In *Proc. CVPR*, 2020.

[20] Y. Li, S. Gu, L. Van Gool, and R. Timofte. Learning filter basis for convolutional neural network compression. In *Proc. ICCV*, pages 5623–5632, 2019.

[21] Y. Li, S. Gu, K. Zhang, L. Van Gool, and R. Timofte. DHP: Differentiable meta pruning via hypernetworks. *arXiv preprint arXiv:2003.13683*, 2020.

[22] Y. Li, K. Zhang, J. Cao, R. Timofte, and L. Van Gool. Localvit: Bringing locality to vision transformers. *arXiv preprint arXiv:2104.05707*, 2021.

[23] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[24] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021.

[25] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, T. K.-T. Cheng, and J. Sun. MetaPruning: Meta learning for automatic neural network channel pruning. In *Proc. ICCV*, 2019.

[26] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. In *Proc. ICLR*, 2019.

[27] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[28] T. Meinhardt, A. Kirillov, L. Leal-Taixe, and C. Feichtenhofer. Trackformer: Multi-object tracking with transformers. *arXiv preprint arXiv:2101.02702*, 2021.

[29] D. Neimark, O. Bar, M. Zohar, and D. Asselmann. Video transformer network. *arXiv preprint arXiv:2102.00719*, 2021.

[30] C. Riquelme, J. Puigcerver, B. Mustafa, M. Neumann, R. Jenatton, A. S. Pinto, D. Keysers, and N. Houlsby. Scaling vision with sparse mixture of experts. *arXiv preprint arXiv:2106.05974*, 2021.

[31] A. Srinivas, T.-Y. Lin, N. Parmar, J. Shlens, P. Abbeel, and A. Vaswani. Bottleneck transformers for visual recognition. *arXiv preprint arXiv:2101.11605*, 2021.

[32] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. *arXiv preprint arXiv:2012.12877*, 2020.

[33] A. Vaswani, P. Ramachandran, A. Srinivas, N. Parmar, B. Hechtman, and J. Shlens. Scaling local self-attention for parameter efficient visual backbones. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12894–12904, 2021.

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[35] T. Verelst and T. Tuytelaars. Dynamic convolutions: Exploiting spatial sparsity for faster inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2320–2329, 2020.

[36] H. Wang, Y. Zhu, B. Green, H. Adam, A. Yuille, and L.-C. Chen. Axial-deeplab: Stand-alone axial-attention for panoptic segmentation. In *Proc. ECCV*, pages 108–126, 2020.

[37] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *arXiv preprint arXiv:2102.12122*, 2021.

[38] L. Wen, D. Du, Z. Cai, Z. Lei, M. Chang, H. Qi, J. Lim, M. Yang, and S. Lyu. UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking. *Computer Vision and Image Understanding*, 2020.

[39] S. Yang, Z. Quan, M. Nie, and W. Yang. Transpose: Towards explainable human pose estimation by transformer. *arXiv preprint arXiv:2012.14214*, 2020.

[40] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE TPAMI*, 38(10):1943–1955, 2015.

[41] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020.

A Appendix

(a) Frame 1 (b) Frame 2 (c) Frame 3 (d) Frame 4

Figure 6: A snippet of 4 frames in UA-DETRAC dataset.

A.1 Redundancy in video streams

The analysis in the main paper identifies the linear layers in DETR as the computation bottleneck. To improve the processing efficiency of the linear layers for video stream processing, we utilize the redundancy in video streams. In Fig. 6, a snippet of 4 video frames is shown, where the background remains stable. In addition, for video object detection, the background area is relatively less important. Thus, for efficient video object detection, the features of the background area could be borrowed from the previous frame. This can save a significant portion of computation and improve the processing efficiency.

A.2 Additional results

In Table 4, we show the result for different number of frames. No gating layers are applied to the first frame. Starting from the second frame, the dynamic gating layers are applied. As the number of frames increase, there is a lightly drop of computational complexity.

Table 4: Results for binary gates when different number of frames are used.

#Frames	γ	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	FLOPs	Params
Baseline		55.9	83.1	64.9	8.6	51.5	75.6	55.3G	45.2M
8	15	55.4	83.1	64.3	9.0	51.2	74.7	29.0G	45.5M
16	15	55.0	82.9	63.7	8.4	50.8	74.2	27.2G	45.5M
24	15	54.8	82.7	63.5	8.3	50.8	73.9	26.8G	45.5M
32	15	54.5	82.5	63.0	8.1	50.5	73.7	26.4G	45.5M