Mobile Vision Transformer-based Visual Object Tracking

Goutam Yelluru Gopal
g_yellur@encs.concordia.ca
Maria A. Amer
amer@ece.concordia.ca

Department of Electrical and Computer Engineering
Concordia University, Montréal
Québec, Canada

Abstract

The introduction of robust backbones, such as Vision Transformers, has improved the performance of object tracking algorithms in recent years. However, these state-of-the-art trackers are computationally expensive since they have a large number of model parameters and rely on specialized hardware (*e.g.*, GPU) for faster inference. On the other hand, recent lightweight trackers are fast but are less accurate, especially on large-scale datasets. We propose a lightweight, accurate, and fast tracking algorithm using Mobile Vision Transformers (MobileViT) as the backbone for the first time. We also present a novel approach of fusing the template and search region representations in the MobileViT backbone, thereby generating superior feature encoding for target localization. The experimental results show that our MobileViT-based Tracker, *MVT*, surpasses the performance of recent lightweight trackers on the large-scale datasets GOT10k and TrackingNet, and with a high inference speed. In addition, our method outperforms the popular DiMP-50 tracker despite having 4.7× fewer model parameters and running at 2.8× its speed on a GPU. The tracker code and models are available at https://github.com/goutamyg/MVT.

1 Introduction

The two prominent paradigms of visual object tracking algorithms are Discriminative Correlation Filters (DCFs) and deep Siamese Networks (SNs) [17]. The DCF-based trackers [1, 9, 14] localize the target object based on the filter response generated by convolving the features extracted from the search region with the filter coefficients learned from the target template. The SN-based trackers [5, 19, 30] perform the cross-correlation (or similar) operation between features extracted from the template and search regions to generate the response map for target localization and bounding box estimation. The explicit learning of target-specific filter coefficients in DCF tracking increases their robustness against semantic background regions compared to SN trackers; however, SNs are faster due to their simpler model architecture supporting end-to-end evaluation on a GPU. With the adoption of powerful backbones and effective feature fusion techniques, SN trackers have shown state-of-the-art performance on various benchmarks [12, 15, 23].

Feature representation of the target object plays a crucial role in tracker performance [28]. Most SN trackers use the ResNet [13] backbone for feature extraction, with ResNet-50 and ResNet-101 being the popular choices. The more recent trackers use pre-trained Vision Transformer (ViT) models [11, 29] as their backbone, surpassing the performance of ResNet-based SN trackers. However, a notable disadvantage of ViT-based trackers is the complexity of their backbone, both in terms of memory (a large number of model parameters) and latency (low inference speed). By deploying these models, achieving high tracking speed on an ordinary CPU or mobile device is challenging. This limitation severely restricts the usage of such tracking algorithms for several resource-constrained applications.

On the other hand, most lightweight tracking algorithms deploy compact convolutional neural network (CNN) backbones to minimize the model latency. The inductive biases of convolutional blocks effectively model the spatially local information related to the target object but fail to capture the global relations essential for accurate target state estimation in tracking [30]. Such a lack of global association between the template and search region in the backbone increases the burden on the feature fusion module (or the neck) to generate the fused encoding favorable for accurate and robust tracking. The self-attention-based Transformers [27] as the backbone is effective at global contextual modeling and have been excellent for tracking [2, 33]; however, they are computationally expensive.

In this paper, we are the first to investigate the usefulness of Mobile Vision Transformers (MobileViTs) as the backbone for single object tracking to present a lightweight but high-performance tracking algorithm, *MVT*. The recent MobileViTs [22] for image classification are known for their low latency, lightweight architecture, and adaptability to downstream tasks, *e.g.*, object detection and semantic segmentation. In addition, while all the related lightweight trackers independently compute the template and search region features in their respective backbone, our *MVT* algorithm employs a hybrid feature extraction method where template and search regions are blended in the backbone by our novel Siamese Mobile Vision Transformer (Siam-MoViT) block.

2 Related Work

Multiple SN-based lightweight trackers have been presented in the last few years. LightTrack [31] employed Neural Architecture Search [6] to present an efficient tracking pipeline. It designed a search space of lightweight building blocks to find the optimal backbone and head architectures with pre-set constraints on the number of model parameters. E.T.Track [2] incorporated Exemplar Transformers for tracking to achieve real-time speed on a CPU. It used a stack of lightweight transformer blocks in the head module to perform target classification and bounding box regression. FEAR [3] tracker deployed a dual-template representation to incorporate temporal information during tracking. With a compact backbone, FEAR achieved over 200 frames-per-second (*fps*) speed on iPhone 11 with negligible impact on battery level. Stark-Lightning [30] used a RepVGG [10] backbone and a transformer-based encoder-decoder architecture in the neck module to model spatio-temporal feature dependencies between the target template and search regions. HiFT [4] proposed a hierarchical feature transformer-based approach for aerial tracking. It generated hierarchical similarity maps from the multi-level convolutional layers in the backbone network to perform a transformer-based fusion of shallow and deep features. SiamHFFT [8] extended the hierarchical feature fusion approach by [4] to model the inter-dependencies within the multi-level features and achieve high tracking speed on a CPU.

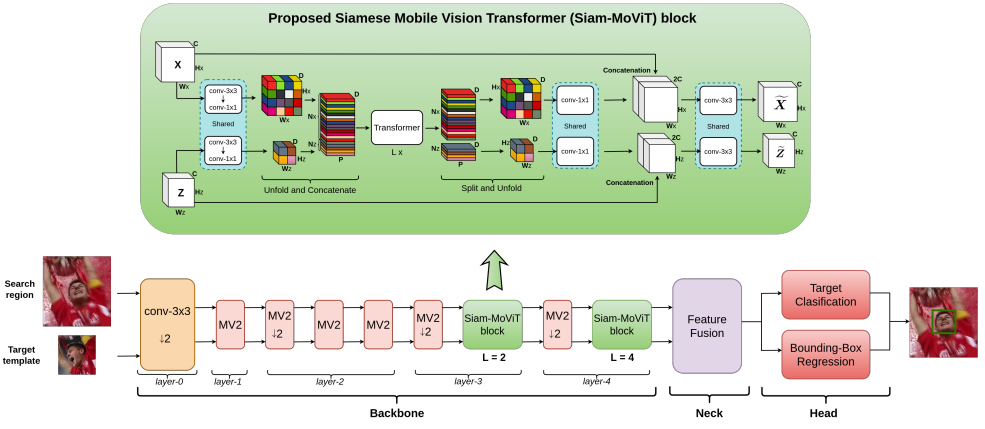

Figure 1: The pipeline of the proposed *MVT* tracker and our Siam-MoViT block. The backbone consists of MobileNetV2 [25] (or MV2) and Siam-MoViT blocks for feature extraction. ↓2 indicates spatial downsampling by a factor of 2. Details of our Siam-MoViT block can be found in Section 3.

Among the related lightweight trackers, LightTrack is closest to our work, having similar neck and head modules but a different backbone. Stark-Lightning uses a transformer-based neck module to fuse features from the template and search regions. In contrast, the proposed *MVT* uses a simple, parameter-free cross-correlation operation in its neck module. E.T.Track uses a transformer-based head module, while our *MVT*'s head module is built using a fully convolutional network. As a post-processing step, the related trackers LightTrack and E.T.Track refine their predicted bounding boxes by penalizing significant changes in bounding box size and aspect ratio between consecutive frames. Unlike these trackers, the proposed *MVT* does not perform such heuristic-based bounding box refinements.

Most importantly, all the related lightweight trackers use a two-stream approach during feature extraction, i.e., the backbone features from the template and search region are computed independently. Such a two-stream computation limits the interaction between the template and search regions to the neck module only, resulting in inferior tracking performance. To alleviate this problem, we propose a hybrid feature extraction method where template and search regions are blended in the backbone by our novel Siam-MoViT block, as shown in Figure 1. The resulting entangled feature representation generated using our Siam-MoViT block improves the tracker performance while maintaining high inference speed. Efficient transformer architectures is an emerging research topic [26] and has been unexplored by previously proposed lightweight trackers. To our knowledge, we are the first to use MobileViT as the backbone for object tracking. We are also the first to propose a tracking pipeline with a joint feature extraction and fusion approach in the tracker backbone.

Our contributions in this paper are thus:

- A novel lightweight tracking algorithm using MobileViTs. We show that the proposed MobileViT-based tracker performs better than related lightweight trackers.

- A hybrid feature extraction approach, intertwining the template and search regions using our Siam-MoViT block, producing better features for target state estimation.

3 Proposed Mobile Vision Transformer-based Tracker

In this section, we discuss the pipeline of our *MVT* algorithm for single object tracking (shown in Figure 1) and information related to model training.

3.1 Proposed *MVT* Backbone and the Siam-MoViT block

The input to our *MVT* backbone is a pair of the target template and search region image patches, $Z_{in} \in R^{W_z \times H_z \times 3}$ and $X_{in} \in R^{W_x \times H_x \times 3}$, respectively. The tracker backbone consists of cascaded MobileNetV2 [25] and the proposed Siam-MoViT blocks, as shown in Figure 1. These modules process the input image patches sequentially, with recurrent spatial downsampling operations to reduce the feature dimensionality. The proposed Siame-MoViT block uses a modified MobileViT block [22], especially around the transformer encoder, to accommodate features from the template and search region.

Our Siam-MoViT block receives a pair of intermediate feature maps Z and X, belonging to the template and search regions, respectively. We assume that Z and X have C channels. Inside the Siam-MoViT block, first, we apply a 3×3 convolutional filter to learn spatially local feature representations. It is followed by a 1×1 convolutional filter, projecting the features onto a D-dimensional space as a linear combination of C input channels. Next, we perform the *unfold and concatenate* operation (*cf.* Figure 1), where we divide the feature maps X and Z into N non-overlapping patches of size $w \times h$. We then flatten these patches to generate tokens of size $P \times N \times D$, where $P = w \cdot h$ and $N = \frac{W \cdot H}{P}$. These tokens are concatenated and passed through a series of L transformer blocks to encode the global relationship between the template and search regions. Our implementation uses the standard multi-headed self-attention transformer encoder blocks [27]. This operation of learning self-attention on the concatenated features facilitates the exchange of information between template and search regions, thereby generating high-quality encodings for robust target localization. To restore the spatial ordering of feature maps, we split the output tokens from the transformer and re-arrange them to obtain feature maps of size $H_z \times W_z \times D$ and $H_x \times W_x \times D$, shown as the *split and unfold* operation in Figure 1. Then, we re-map the number of channels from D to C by applying a 1×1 convolutional filter and concatenate the resulting feature maps with the inputs to the Siam-MoViT block, i.e., Z and X. Finally, we apply a 3×3 convolutional filter on the concatenated feature maps to generate the output of our Siam-MoViT block, denoted as $\tilde{Z}$ and $\tilde{X}$, having the same size as Z and X, respectively. Note that all the MobileNetV2 blocks in the backbone and the CNN blocks within the Siam-MoViT block are applied separately to template and search regions, as shown in Figure 1, with shared weights.

3.2 Neck and Head Modules

The output from the last layer of the *MVT* backbone has feature maps corresponding to the template and search region. We fuse these features in the neck module to generate an encoded feature representation $f_{zx} \in R^{\frac{H_z \cdot W_z}{16^2} \times \frac{H_x}{16} \times \frac{H_x}{16}}$ for target state estimation. For this, we use a simple pointwise cross-correlation operator [32] in the neck module, the same as LightTrack [31]. We use a layer of batch-normalization (BN) [16] before performing cross-correlation. We then apply a 1×1 convolutional *channel-adjust* layer on f_{zx} to match the number of channels between f_{zx} and the head module.

For classification and regression, we adopt the head module from [33], which uses a fully convolutional network (FCN) to perform target classification and bounding box regression.

The FCN consists of a stack of five Conv-BN-ReLU blocks. The classification network predicts a score map $\mathcal{R} \in R^{\frac{H_x}{16} \times \frac{W_x}{16}}$, and the position of the maximum value in $\mathcal{R}$ is considered as the target location. The regressor network predicts the normalized bounding box size (i.e., target width and height) and corresponding local offset values.

3.3 Loss Function for Training

During training, we use loss functions for the classification and regression output by the head module of our *MVT* tracker. As in [33], we use the weighted focal loss L_{cls} to handle the imbalance between positive and negative training examples for target classification. For bounding box regression, same as [33], we use the ℓ_1 and generalized IoU loss functions, denoted by L_1 and L_{giou}, respectively. As in [33], we define the overall loss function as,

$$L_{total} = L_{cls} + \lambda_1 \cdot L_1 + \lambda_2 \cdot L_{giou}, \tag{1}$$

where λ_1 and λ_2 are the hyperparameters controlling the relative impact of L_1 and L_{giou} on the overall training loss.

4 Implementation Details and Experimental Results

This section discusses the implementation details of our *MVT* tracker and compares its results with related lightweight and state-of-the-art heavy trackers. We also discuss the ablation study results for the proposed feature fusion and an attribute-based robustness analysis.

4.1 Implementation Details

We set the dimensions of the inputs to our *MVT* backbone, i.e., Z_{in} and X_{in} from Section 3.1, to 128×128 and 256×256, respectively. We divide our *MVT* backbone into five layers with *layer-id*s for notation convenience, as shown in Figure 1. The number of channels in the feature maps increases along these five layers as $\{3 \rightarrow 16, 16 \rightarrow 32, 32 \rightarrow 64, 64 \rightarrow 96, 96 \rightarrow 128\}$. We set the number of transformer encoders for the proposed Siam-MoViT block in *layer*-3 and *layer*-4 to 2 and 4, respectively. We set the parameters $w = h = 2$ for folding and unfolding operations within our Siam-MoViT block. The number of upscaled channels D in the Siam-MoViT block is set to 144 and 192 for *layer*-3 and *layer*-4, respectively. The backbone has a stride of 16 (i.e., four downsampling operations, each by a factor of two), resulting in feature maps of size 8×8 and 16×16 for the template and search regions, respectively. The *channel-adjust* layer in the neck module, described in Section 3.2, upscales the number of channels from 64 to 256.

We use the training split of the GOT10k dataset [15] to train our model. We use *Adam-W* [21] as the optimizer with a weight decay of 10^{-4}. We trained our model for 100 epochs with 60000 image pairs per epoch, sampled from the training dataset. We use the validation split of GOT10k to compute the values of L_{cls}, L_1, and L_{giou} from Eq. 1 during training to examine the possibility of overfitting. We set the initial learning rate *lr* to 4×10^{-4} and use cosine annealing [20] as the learning rate scheduler (without the warm restarts). We keep the *lr* for the backbone module 0.1 times the *lr* for the rest of the network throughout training. We use the data augmentation techniques horizontal flip and brightness jitter during training. We initialize the backbone using the weights of the pre-trained MobileViT model provided by its authors [22]. Like [22], we do not use positional embeddings for the transformer

blocks in our *MVT* backbone. We set the hyperparameters λ_1 and λ_2 in Eq. 1 to 5 and 2, respectively, as in [33]. We use a single Nvidia Telsa V100 GPU (32GB) for training and set the batch size to 128.

Our choice of optimizer and hyperparameters is based on the training settings typically used by the related trackers. We set our batch size based on the maximum number of images that can be loaded onto the GPU used for training the model. We experimented using the Ray-Tuner package in Pytorch [24] to search for the best set of hyperparameters jointly. First, the hyperparameter search was time-consuming due to the sheer volume. Second, due to a strong inter-dependency between some of the hyperparameters (*e.g.*, batch size and learning rate), it was challenging to find the optimal set using random search-based methods.

During inference, we define the search space at frame t by extracting an image patch around the estimated target location at frame $t-1$, four times the area of the target template. We apply a Hanning window on the classification score map $\mathcal{R}$ as the post-processing step. After this multiplication, we determine the location of the highest value in $\mathcal{R}$ as the target location, and we choose the corresponding bounding box as the tracker output. We define the target annotation from the first frame as the template and do not perform any model update. We generate the GPU-based inference results using an Nvidia RTX 3090 GPU.

4.2 Results and Comparison to Related Work

To demonstrate the effectiveness of the proposed *MVT*, we evaluate it using GOT10k-test [15], TrackingNet-test [23], LaSOT-test [12], and NfS30 [18] datasets. GOT10k has 180 test videos, with non-overlapping target classes from their training videos, to promote generalization during tracker development. TrackingNet has 511 challenging test videos with 15 attributes. GOT10k and TrackingNet datasets sequester the test set annotations and provide an online evaluation server to submit the tracker results to ensure a fair evaluation. LaSOT dataset has 280 test videos, with an average length of 2500 frames per video. NfS dataset has 100 videos captured at 240 and 30 *fps*; we use the 30 *fps* videos. GOT10k provides a train, validation, and test split for its annotated videos, whereas TrackingNet and LaSOT provide the train and test splits. NfS30 has only test videos in the dataset.

GOT10k uses Overlap Ratio (*OR*) and Success Rate (*SR*) at a threshold of 0.5 and 0.75 (i.e., $SR_{0.50}$ and $SR_{0.75}$) to quantify the tracker performance. Metric *OR* is equivalent to Area Under the Curve (*AUC*) [34]. *SR* measures the fraction of frames where the Intersection-over-Union (*IoU*) between groundtruth and predicted boxes is higher than a threshold. TrackingNet uses *AUC*, Precision (*P*), and Normalized-Precison (P_{norm}) for tracker performance. Precision *P* measures the distance between centers of groundtruth and predicted bounding boxes, whereas P_{norm} computes the same metric using normalized bounding boxes. For LaSOT and NfS30, we use the *AUC* and Failure Rate (*FR*) as the performance metrics. *FR* calculates the fraction of frames where the tracker has drifted away, i.e., its bounding box prediction has no overlap with the groundtruth (i.e., *IoU* score is zero).

We compare the results of the proposed *MVT* with the related lightweight trackers: LightTrack [31], Stark-Lightning [30], FEAR-XS [3], and E.T.Track [2], evaluated using the pretrained models provided by their authors. From Table 1, we can see that our *MVT* outperforms all other lightweight trackers on the server-based test set of GOT10k and TrackingNet. No related tracker scores second best constantly for these datasets. On GOT10k-test, our tracker is better by at least 3.7%, 4.6%, and 7.3%, than the second best tracker in terms of *OR*, $SR_{0.50}$, and $SR_{0.75}$, respectively. Recall that GOT10k-test has unseen object classes; this indicates a higher generalization ability of *MVT* towards tracking novel object classes than

Tracker	GOT10k [21] (server)			TrackingNet [22] (server)			NfS30 [23]		LaSOT [24]		*fps*
	$OR\uparrow$	$SR_{0.50}\uparrow$	$SR_{0.75}\uparrow$	$AUC\uparrow$	$P_{norm}\uparrow$	$P\uparrow$	$AUC\uparrow$	$FR\downarrow$	$AUC\uparrow$	$FR\downarrow$	(GPU)
LightTrack [34] (CVPR'21)	0.582	0.668	0.442	72.9	79.3	69.9	0.582	0.146	0.524	**0.116**	99
Stark-Lightning [33] (ICCV'21)	0.596	0.696	0.479	72.7	77.9	67.4	**0.619**	0.111	0.585	0.151	205
FEAR-XS [1] (ECCV'22)	0.573	0.681	0.455	71.5	80.5	69.9	0.487	0.207	0.508	0.273	275
E.T.Track [7] (WACV'23)	0.566	0.646	0.425	74.0	79.8	69.8	0.589	0.172	**0.597**	0.162	53
MVT (ours)	**0.633**	**0.742**	**0.551**	**74.8**	**81.5**	**71.9**	0.603	**0.085**	0.553	0.137	175

Table 1: Comparison of related lightweight SN trackers with our *MVT* on server-based GOT10k-test and TrackingNet-test, and groundtruth available NfS30 and LaSOT-test datasets. The best and second-best results are highlighted in red and blue, respectively.

the related trackers. It also highlights the impact of feature fusion in our tracker backbone compared to other two-stream-based lightweight trackers. We observe a similar behavior using the TrackingNet dataset, where our *MVT* performs better by approximately 2% in AUC, *P*, and P_{norm} than its competitor, LightTrack. No single tracker constantly performs better in *AUC* or *FR* for the NfS30 and LaSOT datasets with groundtruth available for the test sets. For NfS30, our tracker is better by 2.6% in *FR* than the second-best Stark-Lightning while lower by 1.6% in AUC. For LaSOT, our tracker is lower by 2.1% than the second best LightTrack in *FR* and by 4.4% than the best E.T.Track in *AUC*.

Across all the datasets and performance metrics, we can see that our tracker scores the best in most cases (7/10) while being second best in 2/10 cases. Our closest competitor, Stark-Lightning, scores the second-best 5/10 times and the best only once. Regarding speed, our *MVT* runs 175 *fps* during GPU-based evaluation, that is, 15% slower than its competitor Stark-Lightning, as shown in Table 1. It is because Stark-Lightning computes the template region features only once during inference due to its two-stream tracking pipeline. In contrast, our *MVT* requires evaluation of the template features at every frame due to the entanglement of the template and search regions in its backbone, which impacts tracking speed.

4.3 Comparison to State-of-the-art trackers

In Table 2, we compare the proposed *MVT* to state-of-the-art (SOTA) heavyweight trackers on server-based GOT10k and TrackingNet test datasets. We take the values of evaluation metrics for these trackers from the respective papers; however, we compute their *fps* values on a GPU (i.e., Nvidia RTX 3090) and a CPU (i.e., 12th Gen Intel(R) Core-i9 processor), as shown in the last column of Table 2. As we can see, in comparison to the popular DCF-based DiMP-50 [1], the deployment of transformers for feature fusion [5, 30] and as the backbone [7, 33] has improved the tracker performance, but at the cost of increased computational complexity and lowered tracking speed due to higher number of model parameters. In contrast, proposed *MVT* surpasses the performance of the popular DiMP-50 on GOT10k and TrackingNet datasets with 4.7× fewer parameters while running at 2.8× and 2× its speed on a GPU and CPU, respectively. Compared to the best-performing SOTA tracker MixFormer-L [7] in Table 2, proposed *MVT* has 33.43× fewer model parameters and higher *fps*, i.e., 3.87× on GPU and 5.88× on CPU, but has a lower *AUC* of 10.7% on average across the two datasets. Our tracker provides a tradeoff between accuracy and complexity for real-time applications with resource constraints.

4.4 Ablation Study

To analyze the effectiveness of the proposed feature fusion technique deployed in our *MVT* backbone, we evaluate the performance of our tracker trained without the concatenation of

Tracker	GOT10k		TrackingNet		#params ↓	*fps*	
	OR ↑	$SR_{0.50}$ ↑	*AUC* ↑	P_{norm} ↑	(in millions)	GPU ↑	CPU ↑
DiMP-50 [1]	0.611	0.717	74.0	80.1	26.1	61.5	15.0
TransT [5]	0.671	0.768	81.2	85.4	23.0	87.7	2.3
STARK-ST101 [30]	0.688	0.781	82.0	86.9	47.2	80	7.8
OSTrack-384 [33]	0.740	0.835	**83.9**	88.5	92.1	74.4	4.4
MixFormer-L [2]	**0.756**	**0.857**	**83.9**	**88.9**	183.9	45.2	< 5
MVT (ours)	0.633	0.742	74.8	81.5	**5.5**	**175.0**	**29.4**

Table 2: Comparison of our *MVT* with the state-of-the-art heavyweight trackers on server-based GOT10k and TrackingNet test datasets. Best and second best results in accuracy and complexity (i.e., # of parameters and *fps*) are highlighted in red and blue, respectively.

feature fusion in backbone	GOT10k		TrackingNet		NfS30		LaSOT	
	OR ↑	$SR_{0.50}$ ↑	*AUC* ↑	P_{norm} ↑	*AUC* ↑	*FR* ↓	*AUC* ↑	*FR* ↓
✗	0.600	0.703	**74.9**	80.0	0.566	0.122	0.544	0.163
✓(ours)	**0.633**	**0.742**	74.8	**81.5**	**0.603**	**0.085**	**0.553**	**0.137**

Table 3: Ablation study results related to the proposed feature fusion in our *MVT* backbone. Best results are highlighted in red.

the template and search region features inside the proposed Siam-MoViT block (*cf.* Figure 1). Table 3 summarizes the ablation results on the four datasets discussed in Section 4.2. We can see that the proposed feature fusion improves the *OR* (or the equivalent metric *AUC*) by 1.9% on average across all the datasets. It also increases the robustness of our *MVT* tracker by reducing the *FR* on NfS30 and LaSOT datasets by 3.7% and 2.6%, respectively. Learning self-attention on the concatenated features using the transformer blocks in our *MVT* backbone facilitates the global relational modeling *within* and *between* the template and search regions, thereby generating superior features for accurate target localization and robust tracking.

4.5 Robustness Analysis

To analyze the robustness of the proposed *MVT* tracker against various challenging factors (or attributes), we compute its *FR* for attributes annotated under the LaSOT dataset, namely Aspect Ration Change (*ARC*), Background Clutter (*BC*), Camera Motion (*CM*), Deformation (*DEF*), Fast Motion (*FM*), Full Occlusion (*FOC*), Illumination Variation (*IV*), Low Resolution (*LR*), Motion Blur (*MB*), Out-of-View (*OV*), Partial Occlusion (*POC*), Rotation (*ROT*), Scale Variation (*SV*), and Viewpoint Change (*VC*). From Figure 2, we can see that our *MVT* is most robust to target deformation (*DEF*) and appearance changes (*VC*). It is least robust to attribute *FM* since we use a Hanning window on the classification score map during target localization. However, not using the Hanning window deteriorates the robustness of our tracker against *BC* and increases the overall *FR*, as we observed from our experiments. Also, our *MVT* has a higher *FR* for videos under the attribute *LR*. These videos contain small, texture-less target objects such as *volleyball* and *yo-yo*, which are generally fast-moving (i.e., *FM*) and are sensitive to *BC*. SOTA trackers address the challenges of *FM*, *LR*, and *BC* with deep features and larger search area to avoid target loss, but these improvements come at the expense of higher model complexity and memory footprint, as shown in Table 2.

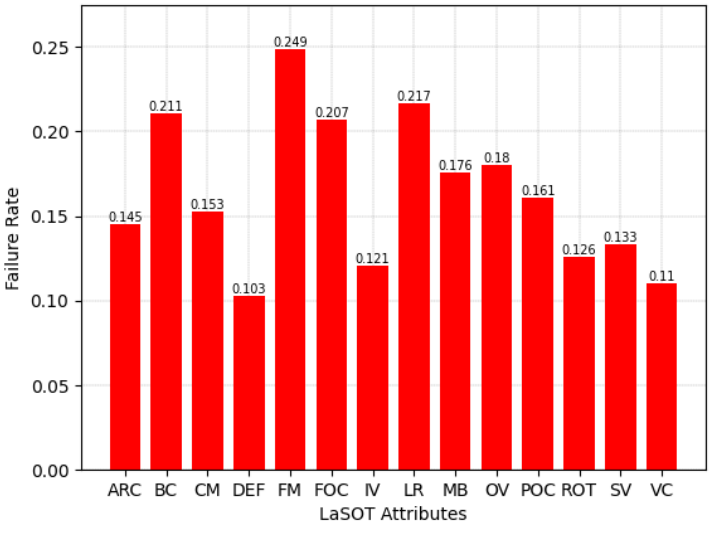

Figure 2: Analyzing the robustness of our *MVT* based on its failure rate FR for different attributes on the LaSOT test dataset. The average FR is 0.137.

5 Conclusion and Future Work

In this paper, we proposed *MVT*, our visual object tracking algorithm that uses, for the first time, the Mobile Vision Transformers as the backbone. We also proposed the Siam-MoViT block to model the global interactions between template and search regions in the tracker backbone, thereby enhancing the quality of feature encodings for target localization. Our simulation results showed that the proposed tracker performed better than the related lightweight trackers on the large-scale GOT10k and TrackingNet datasets, showcasing the effectiveness of the proposed tracking method. Despite having 4.7× fewer model parameters, our *MVT* performs better than the popular DCF-based DiMP-50 tracker, while running at least 2× its speed during CPU and GPU-based evaluation. Our ablation studies highlighted the importance of the proposed feature fusion on our tracker performance.

In our future work, we plan to explore lightweight vision transformer backbone architectures to enhance the quality of encoded features further. Effective feature fusion in the backbone can make the neck module redundant for lightweight tracking, simplifying the tracking pipeline. We also plan to deploy and test our models on low-memory embedded devices, such as smartphones.

References

[1] Goutam Bhat, Martin Danelljan, et al. Learning discriminative model prediction for tracking. In *Proc. IEEE Int. Conf. Computer Vision*, pages 6182–6191, 2019.

[2] Philippe Blatter, Menelaos Kanakis, et al. Efficient visual tracking with exemplar transformers. In *IEEE Winter Conf. App. Computer Vision*, pages 1571–1581, 2023.

[3] Vasyl Borsuk, Roman Vei, et al. FEAR: Fast, efficient, accurate and robust visual tracker. In *Proc. European Conf. Computer Vision*, pages 644–663. Springer, 2022.

[4] Ziang Cao, Changhong Fu, et al. HiFT: Hierarchical feature transformer for aerial tracking. In *Proc. IEEE Int. Conf. Computer Vision*, pages 15457–15466, 2021.

[5] Xin Chen, Bin Yan, et al. Transformer tracking. In *Proc. IEEE Conf. Computer Vision Pattern Recognition*, pages 8126–8135, 2021.

[6] Yukang Chen, Tong Yang, et al. Detnas: Backbone search for object detection. *Advances in Neural Information Processing Systems*, 32, 2019.

[7] Yutao Cui, Cheng Jiang, et al. Mixformer: End-to-end tracking with iterative mixed attention. In *Proc. IEEE Conf. Computer Vision Pattern Recognition*, pages 13608–13618, 2022.

[8] Jiahai Dai, Yunhao Fu, et al. Siamese hierarchical feature fusion transformer for efficient tracking. *Frontiers in Neurorobotics*, 2022.

[9] Martin Danelljan, Goutam Bhat, et al. ECO: Efficient convolution operators for tracking. In *Proc. IEEE Conf. Computer Vision Pattern Recognition*, pages 6638–6646, 2017.

[10] Xiaohan Ding, Xiangyu Zhang, et al. Repvgg: Making vgg-style convnets great again. In *Proc. IEEE Conf. Computer Vision Pattern Recognition*, pages 13733–13742, 2021.

[11] Alexey Dosovitskiy, Lucas Beyer, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

[12] Heng Fan, Hexin Bai, et al. LaSOT: A high-quality large-scale single object tracking benchmark. *Int. J. Computer Vision*, 129:439–461, 2021.

[13] Kaiming He, Xiangyu Zhang, et al. Deep residual learning for image recognition. In *Proc. IEEE Conf. Computer Vision Pattern Recognition*, pages 770–778, 2016.

[14] João F. Henriques, Rui Caseiro, et al. High-speed tracking with kernelized correlation filters. *IEEE Trans. Pattern Anal. Machine Intell.*, 37(3):583–596, 2015. doi: 10.1109/TPAMI.2014.2345390.

[15] Lianghua Huang, Xin Zhao, and Kaiqi Huang. GOT-10k: A large high-diversity benchmark for generic object tracking in the wild. *IEEE Trans. Pattern Anal. Machine Intell.*, 43(5):1562–1577, 2021. doi: 10.1109/TPAMI.2019.2957464.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[17] Sajid Javed, Martin Danelljan, et al. Visual object tracking with discriminative filters and siamese networks: A survey and outlook. *IEEE Trans. Pattern Anal. Machine Intell.*, pages 1–20, 2022. doi: 10.1109/TPAMI.2022.3212594.

[18] Hamed Kiani Galoogahi, Ashton Fagg, et al. Need for speed: A benchmark for higher frame rate object tracking. In *Proc. IEEE Int. Conf. Computer Vision*, pages 1125–1134, 2017.

[19] Liting Lin, Heng Fan, et al. Swintrack: A simple and strong baseline for transformer tracking. *Advances in Neural Information Processing Systems*, 35:16743–16754, 2022.

[20] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Skq89Scxx.

[21] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

[22] Sachin Mehta and Mohammad Rastegari. Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=vh-0sUt8HlG.

[23] Matthias Muller, Adel Bibi, et al. Trackingnet: A large-scale dataset and benchmark for object tracking in the wild. In *Proc. European Conf. Computer Vision*, pages 300–317, 2018.

[24] Adam Paszke, Sam Gross, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

[25] Mark Sandler, Andrew Howard, et al. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proc. IEEE Conf. Computer Vision Pattern Recognition*, pages 4510–4520, 2018.

[26] Yi Tay, Mostafa Dehghani, et al. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28, 2022.

[27] Ashish Vaswani, Noam Shazeer, et al. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

[28] Naiyan Wang, Jianping Shi, et al. Understanding and diagnosing visual tracking systems. In *Proc. IEEE Int. Conf. Computer Vision*, pages 3101–3109. IEEE, 2015.

[29] Haiping Wu, Bin Xiao, et al. CvT: Introducing convolutions to vision transformers. In *Proc. IEEE Int. Conf. Computer Vision*, pages 22–31, 2021.

[30] Bin Yan, Houwen Peng, et al. Learning spatio-temporal transformer for visual tracking. In *Proc. IEEE Int. Conf. Computer Vision*, pages 10448–10457, 2021.

[31] Bin Yan, Houwen Peng, et al. LightTrack: Finding lightweight neural networks for object tracking via one-shot architecture search. In *Proc. IEEE Conf. Computer Vision Pattern Recognition*, pages 15180–15189, 2021.

[32] Bin Yan, Xinyu Zhang, et al. Alpha-refine: Boosting tracking performance by precise bounding box estimation. In *Proc. IEEE Conf. Computer Vision Pattern Recognition*, pages 5289–5298, 2021.

[33] Botao Ye, Hong Chang, et al. Joint feature learning and relation modeling for tracking: A one-stream framework. In *Proc. European Conf. Computer Vision*, pages 341–357. Springer, 2022.

[34] Luka Čehovin, Aleš Leonardis, and Matej Kristan. Visual object tracking performance measures revisited. *IEEE Trans. Image Process.*, 25(3):1261–1274, 2016. doi: 10.1109/TIP.2016.2520370.