General Compression Framework for Efficient Transformer Object Tracking Supplementary Materials

Lingyi Hong[1] Jinglun Li[2] Xinyu Zhou[1] Shilin Yan[1] Pinxue Guo[2] Kaixun Jiang[2] Zhaoyu Chen[2]
Shuyong Gao[1] Runze Li[3] Xingdong Sheng[3] Wei Zhang[1*] Hong Lu[1*] Wenqiang Zhang[1,2*]
[1] Shanghai Key Lab of Intelligent Information Processing,
College of Computer Science and Artificial Intelligence, Fudan University
[2] College of Intelligent Robotics and Advanced Manufacturing, Fudan University
[3] Lenovo Research
honglyhly@gmail.com, wqzhang@fudan.edu.cn

1. Appendix

This appendix is structured as follows:

- In Appendix 1.1, we provide more experiments to verify the strong generalization ability of our CompressTracker.
- In Appendix 1.2, we provide more ablation study results.
- In Appendix 1.3, we show the pseudo code of our CompressTracker.

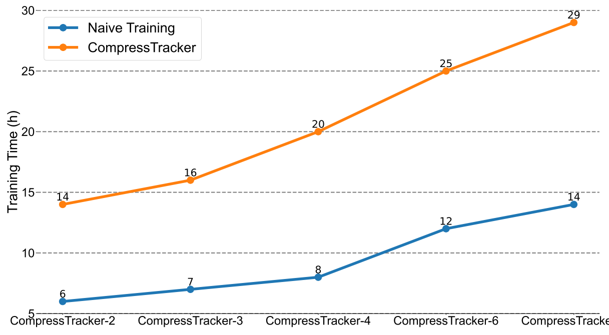

Figure 1. **Training Time.**

1.1. Heterogeneous Structure Robustness

Compressing MixFormerV2. To affirm the generalization ability of our approach, we conduct experiments on MixFormerV2 [1] and SMAT [2]. MixFormerV2-S is a fully transformer tracking model consisting of 4 transformer layers, trained via a complex multi-stages model reduction paradigm. Following MixFormerV2-S, we adopt MixFormerV2-B as teacher and compress it to a student model with 4 layers. The results are shown in Table 1. Our CompressTracker-M-S share the same structure and channel dimension of MLP layers with MixFormerV2-S and outperforms MixFormerV2-S by about 1.4% AUC on LaSOT.

It's worth noting that although CompressTracker-2 and CompressTracker-M-S have similar inference speeds, MixFormerV2-S and CompressTracker-M-S each contain four transformer layers, whereas CompressTracker-2 only has two. The lower number of transformer layers contributes to the slightly lower performance for CompressTracker-2. Additionally, both CompressTracker-4 and CompressTracker-M-S have four transformer layers, but CompressTracker-M-S has a lower hidden feature dim of MLP layer than CompressTracker-4. As highlighted in MixFormerV2-S [1], a reduced feature dimension can lead to decreased accuracy. Consequently, CompressTracker-M-S exhibits slightly lower performance than CompressTracker-4. Moreover, our CompressTracker-4 requires only about 20 hours for training, in contrast to the 120 hours needed for MixFormerV2-S, which also relies on a complex multi-stage training strategy. Besides, the reduction paradigm in MixFormerV2 limits the student model's structure, while our framework supports a diverse range of transformer architectures thanks to our stage division.

Compressing OSTrack to SMAT. SMAT replace the vanilla attention in transformer layer with separated attention. We compress OSTrack into a student model CompressTracker-SMAT, aligning the number and structure of transformer layer with SAMT. We maintain the decoder of OSTrack for CompressTracker-SMAT. CompressTracker-SMAT surpasses SMAT by 1.1% AUC on LaSOT, which demonstrates that our framework is flexible and not limited by the structure of transformer layer.

Based on results in Table 1 and 2 in main paper and Table 1 and 2, our CompressTracker achieves an optimal trade-off between efficiency and performance and is applicable to any *teacher model*, any *resolution*, any *stage number*,

*Corresponding Author

Method	LaSOT			LaSOT$_{ext}$		TNL2K		TrackingNet			UAV123		FPS
	AUC	P$_{Norm}$	P	AUC	P	AUC	P	AUC	P$_{Norm}$	P	AUC	P	
MixFormerV2-B [1]	70.6	80.8	76.2	50.6	56.9	57.4	58.4	83.4	88.1	81.6	69.9	92.1	165
MixFormerV2-S [1]	60.6	69.9	60.4	43.6	46.2	48.3	43.0	75.8	81.1	70.4	65.8	86.8	325
CompressTracker-M-S	**62.0** $_{88\%}$	**70.9**	**63.2**	**44.5** $_{88\%}$	**47.1**	**50.2** $_{87\%}$	**47.8**	**77.7** $_{93\%}$	**82.5**	**73.0**	**66.9** $_{96\%}$	**87.1**	**325** $_{1.97\times}$

Table 1. **Compress MixFormerV2.** We compress MixFormerV2 into CompressTracker-M-S with 4 layers, which is the same as MixFormerV2-S including *the dimension of MLP layer*. We report the performance on 5 benchmarks and calculate the performance gap in comparison to the origin MixFormerV2-B. Our CompressTracker-M-S outperforms MixFormerV2-S under the same setting.

Method	LaSOT			LaSOT$_{ext}$		TNL2K		TrackingNet			UAV123		FPS
	AUC	P$_{Norm}$	P	AUC	P	AUC	P	AUC	P$_{Norm}$	P	AUC	P	
OSTrack-256 [3]	69.1	78.7	75.2	47.4	53.3	54.3	-	83.1	87.8	82.0	68.3	-	105
SMAT [2]	61.7	71.1	64.6	-	-	-	-	78.6	84.2	75.6	64.3	83.9	**158**
CompressTracker-SMAT	**62.8** $_{91\%}$	**72.2**	**64.0**	**43.4** $_{92\%}$	**46.0**	**49.6** $_{91\%}$	**46.9**	**79.7** $_{96\%}$	**85.0**	**75.4**	**65.9** $_{96\%}$	**86.4**	138 $_{1.31\times}$

Table 2. **Compress OSTrack for SMAT.** We compress OSTrack into CompressTracker-SAMT with 4 SMAT layers, which is the same as SMAT. We report the performance on 5 benchmarks and calculate the performance gap in comparison to the original OSTrack. Our CompressTracker-SAMT outperforms SMAT under the same setting.

any *teacher model size*, and any *student architectures*, which highlights the superiority and strong generalization ability of our CompressTracker.

1.2. More Ablation Study

We represent more ablation studies on LaSOT to explore the factors contributing to effectiveness of our CompressTracker. Unless otherwise specified, teacher model is OSTrack,and student model has 4 encoder layers. The student model is trained for 300 epochs, and the p_{init} is set as 0.5.

Training Time. We compare the training time of CompressTracker with 500 training epochs across different layers in Figure 1. 'Naive Training' denotes solely training on groundtruth data with 300 epochs, and 'CompressTracker' represents our proposed training strategy with 500 epochs. The training time is recorded on 8 NVIDIA RTX 3090 GPUs. Although our CompressTracker requires a longer training time compared to the 'Naive Training', the increased computational overhead remains within acceptable limits.

1.3. Replacement Training

We present the pseudocode for the training and testing phases of CompressTracker in Algorithm 2 and Algorithm 3, respectively. Additionally, the pseudocode of OSTrack [3] is also shown in Algorithm 1. During training process, we employ Bernoulli sampling to implement a replacement training strategy, while in the test phase, we integrate the student layers and discard the teacher layer.

Algorithm 1 Pseudocode of OSTrack in a PyTorch-like style

```
# z/x: RGB image of template/search
    region
# patch_embed: patch embedding layer,
# pos_embed_z/pos_embed_z: position
    embedding for template/search region
# blocks: transformer block layers
# decoder: decoder network

def forward(x, z):
    # patch embedding layer
    x, z = patch_embed(x), patch_embed(z
        )

    # add position embedding
    x, z = x + pos_embed_x, z +
        pos_embed_z

    # concat
    x = torch.cat([z, x], dim=1)

    # transformer layers
    for i, blk in enumerate(blocks):
        x = blk(x)

    # decode the matching result
    x = decoder(x)
```

Algorithm 2 Pseudocode of CompressTracker for Training in a PyTorch-like style

```
# z/x: RGB image of template/search
   region
# patch_embed: patch embedding layer,
# pos_embed_z/pos_embed_z: position
   embedding for template/search region
# bernoulli_sample: bernoulli sampling
   function with probability of p
# n_s/n_t: layer number of student/
   teacher model
# teacher_blocks: transformer block
   layers of a pretrained teacher
# student_blocks: transformer block
   layers of student model
# decoder: decoder network

def forward(x, z):
   # patch embedding layer
   x, z = patch_embed(x), patch_embed(z
      )

   # add position embedding
   x, z = x + pos_embed_x, z +
      pos_embed_z

   # concat
   x = torch.cat([z, x], dim=1)

   # replacement sampling
   inference_blocks = []
   for i in range(n):
      if bernoulli_sample() == 1:
         inference_blocks.append(
            student_blocks[i])
      else:
         for j in range(n_t//n_s):
            inference_blocks.append(
               teacher_blocks[i*(n_t//
               n_s) + j])

   # randomly replaced transformer
      layers
   for i, blk in enumerate(
      inference_blocks):
      x = blk(x)

   # decode the matching result
   x = decoder(x)
```

Algorithm 3 Pseudocode of CompressTracker for Testing in a PyTorch-like style

```
# z/x: RGB image of template/search
   region
# patch_embed: patch embedding layer,
# pos_embed_z/pos_embed_z: position
   embedding for template/search region
# student_blocks: transformer block
   layers of student model
# decoder: decoder network

def forward(x, z):
   # patch embedding layer
   x, z = patch_embed(x), patch_embed(z
      )

   # add position embedding
   x, z = x + pos_embed_x, z +
      pos_embed_z

   # concat
   x = torch.cat([z, x], dim=1)

   # transformer layers
   for i, blk in enumerate(
      student_blocks):
      x = blk(x)

   # decode the matching result
   x = decoder(x)
```

References

[1] Yutao Cui, Tianhui Song, Gangshan Wu, and Limin Wang. Mixformerv2: Efficient fully transformer tracking. *Advances in Neural Information Processing Systems*, 36, 2024. 1, 2

[2] Goutam Yelluru Gopal and Maria A Amer. Separable self and mixed attention transformers for efficient object tracking. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 6708–6717, 2024. 1, 2

[3] Botao Ye, Hong Chang, Bingpeng Ma, Shiguang Shan, and Xilin Chen. Joint feature learning and relation modeling for tracking: A one-stream framework. In *European conference on computer vision*, pages 341–357. Springer, 2022. 2