Online Temporal Action Localization with Memory-Augmented Transformer

Supplementary Material

Youngkil Song* Dongkeun Kim* Minsu Cho Suha Kwak

Pohang University of Science and Technology (POSTECH), South Korea
{songyk,kdk1563,mscho,suha.kwak}@postech.ac.kr
https://cvlab.postech.ac.kr/research/MATR/

This supplementary material provides contents omitted in the main paper due to the page limit. In Sec. A, we provide further explanation regarding the 2D temporal positional encoding used in the start decoder of the instance decoding module. Sec. B presents the memory queue update procedure. In Sec. C, we analyze AP scores for each class in the THUMOS14 and MUSES datasets. Sec. D describes additional experimental details including hyperparameter settings. Sec. E presents in-depth analysis of our framework including the effects of the hyperparameters and the post-processing. Sec. F gives more qualitative results of our model on the THUMOS14 dataset.

A About 2D Temporal Positional Encoding

In this section, we provide a more detailed explanation regarding the implementation of the 2D temporal positional encoding, which is applied to the *key* for the cross-attention layers of the start decoder in the instance decoding module. As explained in Sec. 3.3, the *key* for the cross-attention layers consists of the memory queue concatenated with the current segment features. Therefore, the 2D temporal positional encoding should encompasses the relative position of segments within the memory queue and the relative position of the frames within the segment. Note that the relative position is defined with respect to the current frame and the position of the current segment features it belongs to.

To be specific, we explain a practical example of illustrating the application of the 2D temporal positional encoding. Consider a scenario where the memory queue stores 4 segment features, and each segment is composed of 4 frames. If the indices of the segments stored in the memory queue are 3, 4, 9, and 10, with the current segment indexed as 13, then the indices for the memory features of the start decoder are 3, 4, 9, 10 and 13. In this case, the relative segment positions from the current segment are 10, 9, 4, 3, and 0, respectively, while the relative frame positions within each segment are 3, 2, 1, and 0. These two relative positions are encoded using sinusoidal positional encoding of dimension $D/2$ each, then concatenated to obtain the D-dimensional positional encoding. By separating the relative segment positions from relative frame positions, the 2D

* Equal contribution.

Algorithm 1: Pseudocode for memory update.

```
1  class UpdateMemQueue():
2      self.mem_queue = [] # cached memory queue
3      self.max_len # max memory len
4
5      def forward(self, segment, flag): # update memory queue
6          if flag == True:
7              self.mem_queue.append(segment)
8          if len(self.mem_queue) > self.max_len:
9              self.mem_queue.pop_first()
10
11 if training == True: # select flag type
12     flag = gt_flag
13 else:
14     flag_prob = flag_token.sigmoid()
15     if flag_prob > flag_threshold:
16         flag = True
17     else:
18         flag = False
19
20 if isNewVideo: # reset update module when a new video starts
21     update_mem = UpdateMemQueue()
22
23 update_mem(segment, flag) # update memory queue
```

temporal positional encoding becomes more efficient in representing positional encodings for a large number of frames.

B Algorithm of Memory Update Module

We present the details of memory update procedure in Algorithm 1. The algorithm provides pseudocode of an "UpdateMemQueue" function and its usage in the memory queue update procedure. The input parameters of the function are `segment` and `flag` (line 23), where `segment` refers to the current segment features and `flag` is the Boolean flag to determine whether the current segment should be stored in the memory queue or not. During training, the ground-truth flag is used (line 11-12). During inference, the flag is true if the flag probability exceeds the flag threshold, and false otherwise (line 13-18). Note that the function is reinitialized at the start of each new video to prevent the mixing of the memory queues across different videos (line 20-21).

C Analysis of AP scores of MATR

Fig. S1 and Fig. S2 show the Average Precision (AP) scores of MATR on the THUMOS14 [2] and MUSES dataset [6]. In the THUMOS14 dataset, action

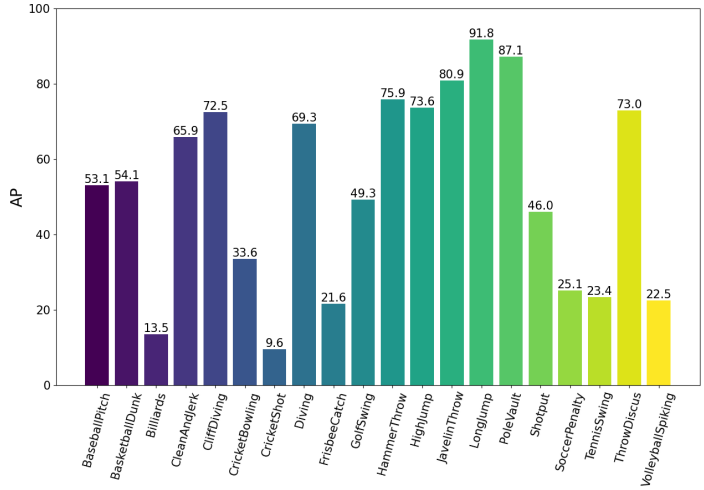

Fig. S1: The Average Precision (AP) for each class at IoU threshold = 0.5 on the THUMOS14 dataset.

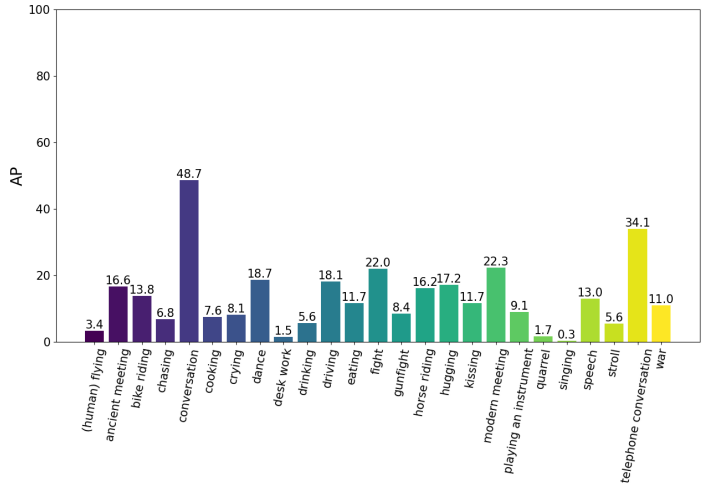

Fig. S2: The Average Precision (AP) for each class at IoU threshold = 0.5 on the MUSES dataset.

classes *Long Jump*, *Pole Vault*, and *Javelin Throw* show the highest performance, while action classes *Cricket Shot* and *Billiards* show the lowest AP. In the MUSES dataset, the *Conversation* class has the highest AP of 48.7, while *Singing* is measured with the lowest AP of 0.3. Both datasets encompass human-centric action classes, resulting in small differences between action classes and making it challenging to classify different action classes, requiring fine-grained classification. Specifically, the MUSES dataset has confusing action classes such as *quarrel* and *fight*, *conversation* and *telephone conversation*.

D Experimental Details

Hyperparameters. We use a frozen two-stream TSN [9] pretrained on Kinetics [1] to extract RGB and flow features for THUMOS14, and an I3D [1] trained on Kinetics [1] utilizing only RGB features for MUSES, following previous On-TAL work [3, 4, 8]. The segment size L_s is set to 64 in THUMOS14 and 75 in MUSES. The dimension of the segment feature is set to $D = 1024$ for both datasets. For the memory-augmented video encoder, we stack 3 Transformer layers with 8 attention heads for both datasets. The flag threshold θ is set to 0.5. For the instance decoding module, we stack 5 Transformer layers with 4 attention heads for both datasets. The number of the class query and the boundary query $N = 10$ for THUMOS14 and $N = 6$ for MUSES. We set the online NMS threshold used for post-processing to 0.3. The memory size L_m is set to 7 for THUMOS14, while 15 for MUSES. For the prediction heads, T_d and T_a are set to 16.

Training. We train our model with Adam optimizer [5] with $\beta_1 = 0.9, \beta_2 = 0.999$, and $\epsilon = 1\text{e}{-8}$ for 100 epochs on the THUMOS14 dataset and 30 epochs on the MUSES dataset. We use the CosineAnnealing scheduler [7] with warmup restarts which have the minimum learning rate of 1e−8, the maximum learning rate of 1e−5, $T_{\text{cycle}} = 10, T_{\text{up}} = 3$, and $\gamma = 0.9$, where T_{cycle} is the number of epochs of one learning cycle, T_{up} is the number of up-scaling epochs of one cycle, and γ is the decreasing ratio of the lr_{max} that decreases with each cycle. The batch size is set to 64 for THUMOS14, while 75 for MUSES. The focal loss coefficient is $\alpha = 0.25$ and $\gamma = 2$ for THUMOS14 and $\alpha = 1$ and $\gamma = 5$ for MUSES. All loss coefficients are set to 1.

E Additional Experiments

Analysis of the number of instance queries. In Table S1, we investigate the effects of the number of instance queries N in the instance decoding module. The performance tends to decrease with the number of instance queries in general, the optimal result is gained when $N = 10$. Note that 10 is the smallest possible N, since N should be greater than or equal to the maximum number of action instances that appear within the specified range $[t - T_d + 1, t + T_a]$, where T_d refers to the size of detection regions before the current timestamp t and T_a refers to the size of anticipation regions after the current timestamp t.

Table S1: Effects of the number of the instance queries N of the instance decoding module on the THUMOS14 dataset.

N	0.3	0.4	0.5	0.6	0.7	Average
10 (Ours)	**70.3**	**62.7**	**52.1**	**38.6**	**23.7**	**49.5**
20	66.6	61.0	51.4	37.3	23.1	47.9
30	64.8	59.4	48.8	35.9	21.1	46.0

Table S2: Performance analysis according to the NMS threshold on the THUMOS14 dataset.

NMS threshold	0.3	0.4	0.5	0.6	0.7	Average
0.1	68.7	61.1	50.4	37.3	23.7	48.2
0.2	69.8	61.9	51.3	37.9	23.4	48.9
0.3 (Ours)	**70.3**	**62.7**	52.1	38.6	23.7	**49.5**
0.4	69.6	62.3	**52.3**	38.5	23.4	49.2
0.5	67.8	61.0	51.6	38.6	23.7	48.5
0.6	66.2	60.0	51.0	**38.9**	**24.6**	48.1
0.7	62.6	57.0	49.0	37.9	24.4	46.2

Analysis of the post-processing. Table S2 summarizes the effects of the NMS post-processing. As explained in Sec. 3.4 of the main paper, we adopt two NMS post-processing for MATR. One is NMS among the action proposals generated at the current timestamp, and the other is NMS among the highly overlapped action proposals generated from the past. The NMS threshold in Table S2 represents the tIoU threshold used by the both NMS post-processing. Action instances surpassing the tIoU threshold are considered as the same instances and removed. Our model performs robustly across various NMS thresholds, the optimal result is obtained when the threshold is 0.3.

Analysis of the T_d and T_a. As presented in Table S3, we show the effects of the hyperparameters T_d and T_a, where T_d refers to the size of detection regions before the current timestamp t and T_a refers to the size of anticipation regions after the current timestamp t. As described in the Sec. 3.5 of the main paper, MATR is trained to predict action instances with action end time is in the range $[t - T_d + 1, t + T_a]$. Comparing A with B and D, E with F, even with the same total range length $T_d + T_a$, predicting action instances that end after the current timestamp ($e_m > t$) is very effective. In the comparison between B, C, and D with the same T_d, mAP increases as T_a increases. The best performance is achieved when both T_d and T_a is set to 16. Therefore, we adopt the hyperparameter setting D for the final model.

F More Qualitative Results

Fig. S4 visualizes the predictions of MATR and OAT-OSN [4]. The results demonstrate that MATR is able to localize action instances better than existing On-TAL methods. In Fig. S4 (a) and (b), MATR accurately predicting the

Table S3: Analysis of the detection region size T_d and the anticipation region size T_a on the THUMOS14 dataset.

Method	T_d	T_a	0.3	0.4	0.5	0.6	0.7	Average
A	8	8	67.5	62.1	**52.1**	38.5	23.1	48.7
B	16	0	60.5	54.1	41.5	29.5	15.4	40.2
C	16	8	67.4	61.8	51.0	37.8	24.1	48.4
D (Ours)	16	16	**70.3**	**62.7**	**52.1**	**38.6**	23.7	**49.5**
E	24	8	67.3	60.9	51.7	39.1	**24.6**	48.7
F	32	0	63.2	54.9	43.1	28.9	17.4	41.5

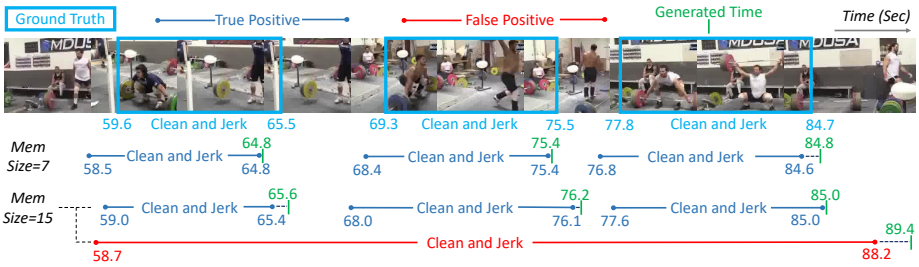

Fig. S3: Qualitative results of MATR with different memory sizes on the THUMOS14.

action end time by avoiding uncertain future predictions since MATR removes action instances where the predicted end time is beyond the current timestamp. In Fig. S4 (c), MATR predicts instances of different classes having similar time intervals, such as *Cliff Diving* and *Diving*. In Fig. S4 (d), MATR detects the second *Frisbee Catch* action, which is challenging to detect solely based on the visual features from the current segment, by leveraging information regarding the previous *Frisbee Catch* action instances stored in the memory queue as long-term context. In Fig. S4 (e), MATR operates robustly even on instances that are prone to being segmented into multiple parts.

Limitations As shown in Fig. S3, when the memory queue contains multiple action instances, there is a risk of action instances being matched with incorrect start points. When the memory sizes are both 7 and 15, the model predicts accurate actions and intervals for all three ground-truth instances. However, with a memory size of 15, the memory queue contain more start timestamps with similar appearances, leading to additional instance being incorrectly matched with incorrect start timestamp.

Fig. S4: More qualitative results on the THUMOS14 dataset. Generated time is the timestamp when the predicted instance is generated.

References

1. Carreira, J., Zisserman, A.: Quo vadis, action recognition? a new model and the kinetics dataset. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
2. Jiang, Y.G., Liu, J., Roshan Zamir, A., Toderici, G., Laptev, I., Shah, M., Sukthankar, R.: THUMOS challenge: Action recognition with a large number of classes. http://crcv.ucf.edu/THUMOS14/ (2014)
3. Kang, H., Kim, K., Ko, Y., Kim, S.J.: Cag-qil: Context-aware actionness grouping via q imitation learning for online temporal action localization. In: Proc. IEEE International Conference on Computer Vision (ICCV). pp. 13729–13738 (2021)
4. Kim, Y.H., Kang, H., Kim, S.J.: A sliding window scheme for online temporal action localization. In: Proc. European Conference on Computer Vision (ECCV). pp. 653–669. Springer (2022)
5. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Proc. International Conference on Learning Representations (ICLR) (2015)
6. Liu, X., Hu, Y., Bai, S., Ding, F., Bai, X., Torr, P.H.: Multi-shot temporal event localization: a benchmark. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 12596–12606 (2021)
7. Loshchilov, I., Hutter, F.: Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983 (2016)
8. Tang, T.N., Park, J., Kim, K., Sohn, K.: Simon: A simple framework for online temporal action localization. arXiv preprint arXiv:2211.04905 (2022)
9. Wang, L., Xiong, Y., Wang, Z., Qiao, Y., Lin, D., Tang, X., Van Gool, L.: Temporal segment networks: Towards good practices for deep action recognition. In: Proc. European Conference on Computer Vision (ECCV). pp. 20–36. Springer (2016)