ETAD: Training Action Detection End to End on a Laptop
Supplementary Material

Shuming Liu[1], Mengmeng Xu[1], Chen Zhao[1], Xu Zhao[2], Bernard Ghanem[1]
[1]King Abdullah University of Science and Technology [2]Shanghai Jiao Tong University
{shuming.liu, mengmeng.xu, chen.zhao, bernard.ghanem}@kaust.edu.sa zhaoxu@sjtu.edu.cn

A. Detailed Architecture of ETAD

In this section, we describe the detailed designs of two modules in ETAD: feature enhancement module and proposal evaluation module. Then, we introduce the loss function of our model, and more implementation details.

A.1. Feature Enhancement Module

As shown in Fig. 1, feature enhancement module adopts two LSTM [3] with different aggregation directions to capture both forward and backward context. The residual connection in the middle can mitigate the effect of forgetting issue brought by the LSTM. Group normalization layer with group number 16 and ReLU are used after each convolution layer. We also study its effectiveness by replacing it with a convolutional network (Conv) and a vanilla transformer. In Tab. 1 (top), Transformer shows the lowest mAP, since it requires more data to converge and stronger regularization to optimize. Conv also shows low mAP due to its inability to capture long-range context. Compared with the above two, our LSTM-based module shows the best performance.

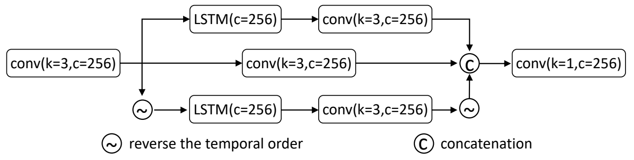


Figure 1. **Architecture of feature enhancement module.** K, C stand for kernel size and channel number of corresponding layer.

A.2. Proposal Evaluation Module

The architecture of proposal evaluation module is shown in Tab. 2. Given a candidate proposal set, we first use the interpolation and rescaling algorithm in G-TAD [8] as RoI alignment to extract the proposal features. Then we refine the proposals with several FC layers from three aspects: (1) The offset of the predicted start/end boundary. (2) The offset of the predicted center/width. (3) The IoU score between the proposal with the ground truth. What's more, we find adding a branch to classify the proposal startness and endness is helpful for IoU regression. After one proposal evaluation module, proposals will be refined by the average of start/end offset and center/width offset.

Table 1. **Ablation of different feature enhancement modules (Feat. Layer), and different number of proposal evaluation modules (#PEM)** on ActivityNet-1.3 with TSM-R50.

Feat. Layer	#PEM	0.5	0.75	0.9	avg.
Transformer	1	52.75	36.34	7.28	35.34
Conv	1	53.06	36.72	7.32	35.71
LSTM	1	53.52	37.54	6.08	**36.10**
LSTM	2	54.02	37.84	7.96	36.59
LSTM	3	53.79	37.59	10.56	**36.79**
LSTM	4	53.26	37.65	9.65	36.57

To further improve the boundary precision of predicted actions, we follow the cascade-RCNN [1] to stack three proposal evaluation modules, where the proposals generated by the first stage are further refined in the second stage and so forth. We use the increased IoU thresholds for the three stages, namely 0.7, 0.8, and 0.9. As such, the proposal boundaries are expected to become more accurate after each stage, which is also proved in Tab. 1. It is worth mentioning that our cascade proposal refinement does not rely on an additional proposal generation network, which is different from [6].

A.3. Loss function.

The loss function of our method consists of boundary evaluation loss and cascade proposal refinement loss. $\mathcal{L}$ is computed as follows:

$$\mathcal{L} = \mathcal{L}_{ce:bd_s} + \sum_{i=1,2,3} \left(\mathcal{L}^i_{ce:bd_p} + \mathcal{L}^i_{iou} + \lambda \mathcal{L}^i_{rg:secw} \right) \quad (1)$$

where i is the index of the cascade proposal evaluation module, and weight λ is set to 10 for balancing the losses.

Table 2. **The detailed architecture of proposal evaluation module.** N is the number of candidate action proposals.

Proposal Start/End Offset Regression			
layer	dim	act	output size
proposal start/end feature			$128 \times 8 \times N$
FC	512	$relu$	$512 \times N$
FC	128	$relu$	$128 \times N$
FC	128	$relu$	$128 \times N$
FC	128	$relu$	$128 \times N$
FC	1	$\times$	$1 \times N$

Proposal Center/Width Regression			
layer	dim	act	output size
proposal extended feature			$128 \times 32 \times N$
FC	512	$relu$	$512 \times N$
FC	128	$relu$	$128 \times N$
FC	128	$relu$	$128 \times N$
FC	128	$relu$	$128 \times N$
FC	2	$\times$	$2 \times N$

Proposal IoU Regression			
layer	dim	act	output size
proposal extended feature			$128 \times 32 \times N$
FC	512	$relu$	$512 \times N$
FC	128	$relu$	$128 \times N$
FC	128	$relu$	$128 \times N$
FC	128	$relu$	$128 \times N$
FC	2	$sigmoid$	$2 \times N$

$\mathcal{L}_{ce:bd_s}$ in the boundary evaluation module uses batch-level positive-negative-balanced cross entropy to supervise the startness or endness of each snippet, which is the same as proposed in BMN [5]. We use the same loss for $\mathcal{L}_{ce:bd_p}$ in proposal evaluation module to compute the cross entropy of the proposal's startness and endness. Using $\mathcal{L}_{ce:bd_p}$ is helpful for stabilizing the learning of IoU confidence. $\mathcal{L}_{iou}$ contains a classification loss and a regression loss for the predicted IoU, which follows [5]. The classification loss is cross-entropy loss, and the regression loss is L2 loss. For $\mathcal{L}_{rg:secw}$, we use the smooth-L1 loss for regressing start/end offset and center/width offset. We only do regression on positive samples, and the threshold of positive samples is gradually improved in the cascaded proposal evaluation module, *i.e.* 0.7, 0.8, 0.9.

A.4. Implementation Details

Training. In ActivityNet-1.3, we resize the feature sequences to a fixed length of 128 snippets. For THUMOS-14, we sample the features per 4 frames with fps 30, and utilize the sliding window approach with window length 128 and stride 64 for videos to generate training samples.

Inference. To post-process network outputs, we use the boundary selecting method in [5] to select proposals with high startness and endness, and use the averaged proposal boundary generated from three proposal evaluation modules. Soft-NMS is adopted based on proposal confidence scores $p = p_s \cdot p_e \cdot p_{iou}$, where p_s and p_e are from $\mathcal{L}_{ce:bd_s}$ standing for the start and end probabilities of a proposal, and p_{iou} is the IoU score of the proposal from $\mathcal{L}_{iou}$.

B. Effectiveness of APS

B.1. APS in End-to-end training

In Fig.4 of the main paper, we compared the performance with different APS ratios given pre-extracted features. Here, we also evaluate our method with different APS ratios under end-to-end training. As shown in Tab. 3, end-to-end training generally improves the detection performance if the APS ratio is larger than 2%, and the performance also starts to saturate with larger ratios. However, if the APS ratio is larger than 10%, the mAP becomes lower. This is because too many proposals would cause the learning bias of the training dataset (*e.g.* large proposals in ActivityNet). Thus, we choose 6% as the APS ratio by default.

Table 3. **Ablations of different APS ratio with end-to-end training** on ActivityNet-1.3 with TSM-R50.

E2E	0.1%	0.2%	2%	4%	**6%**	10%	20%	100%
✗	34.59	35.22	35.70	35.93	**36.13**	36.09	36.07	36.10
✓	34.50	35.21	36.34	36.41	**36.79**	36.72	36.66	36.51

B.2. APS during inference

As the default, ETAD only performs APS during training to reduce the computation cost. In inference, we use all the predicted proposals for higher detection performance. However, as a tool for selecting proposals, APS can also be applied during inference. Based on such motivation, we adopt grid sampling strategy with APS during inference, and Tab. 4 shows that APS is also effective for reducing inference complexity while preserving accuracy. Only the sampling ratio is smaller than 10%, the mAP starts to decrease visibly. We did not conduct APS in inference as the default, considering that the impact of different APS ratios is rather small for the inference time and the inference GFLOPs in end-to-end setting.

Table 4. **Ablations of different APS ratios during inference** on ActivityNet-1.3 with TSM-R50.

APS Ratio	**100%**	20%	15%	10%	6%
mAP	**36.79**	36.77	36.74	36.71	36.51

Table 5. **Comparison of ETAD with other state-of-the-art methods on HACS with same pre-extracted SlowFast features.** Total GPU memory with batch size 16 is reported.

Methods	0.5	0.75	0.95	Avg. mAP	Memory (GB)	Training Time
BMN [5]	52.49	36.38	10.37	35.76	12.10	58 min
BMN [5] + TCANet [6]	55.60	40.01	11.47	38.71	12.34	104 min
ETAD (random)	55.71	39.06	13.78	**38.77**	**3.28**	**50 min**
ETAD (grid)	55.49	39.09	14.08	38.76	3.28	50 min
ETAD (block)	51.46	34.26	11.43	34.49	3.28	50 min

Table 6. **Comparison of different gradient sampling ratio γ on THUMOS test set.** The GPU memory is reported by each video.

Feature Encoder	0.3	0.4	0.5	0.6	0.7	Avg. mAP	Memory (GB)	Training Time
SlowOnly ($\gamma = 0\%$)	52.45	44.11	34.32	24.84	15.89	34.32	-	-
SlowOnly ($\gamma = 10\%$)	59.72	52.74	42.73	32.98	23.02	42.23	1.06	2.08 h
SlowOnly ($\gamma = 30\%$)	60.66	52.87	42.95	33.31	23.38	42.63	1.06	2.25 h
SlowOnly ($\gamma = 100\%$)	60.18	52.93	44.40	33.88	23.76	43.03	1.06	3.09 h
TSM ($\gamma = 0\%$)	52.18	42.80	33.10	24.20	14.05	33.26	-	-
TSM ($\gamma = 10\%$)	57.63	48.76	38.12	28.55	18.39	38.28	1.19	2.23 h
TSM ($\gamma = 30\%$)	56.50	49.16	39.17	29.47	19.07	38.67	1.19	2.51 h
TSM ($\gamma = 100\%$)	57.44	48.99	39.55	29.37	18.60	38.79	1.19	3.92 h

C. Results on HACS dataset

We also report the results of ETAD on HACS [9] dataset based on the pre-extracted SlowFast feature, since this is a fair comparison with other state-of-the-art methods that use the same feature. HACS is a recent large-scale temporal action localization dataset, containing 140K action instances from 50K videos including 200 action categories. In this dataset, we adopt SlowFast [2] features provided by [6] and rescale the feature sequences to 224 snippets. The only training difference from ActivityNet is that we use the learning rate of 4×10^{-4} and batch size of 16 on HACS.

As shown in Tab. 5, ETAD can outperform the baseline method BMN [5] by a large margin. Compared with state-of-the-art method TCANet [6], ETAD can also achieve comparable performance. (1) Particularly, the training time is visibly reduced from 104 mins to 50 mins, and the GPU memory decreases from 12.34 GB to 3.28 GB. This further proves the existence of proposal redundancy in TAD and the effectiveness of our APS design. (2) ETAD also exceeds TCANet on the high IoU threshold scenario by 2.5%, which is similar to ActivityNet-1.3 and THUMOS14. (3) What's more, TCANet relays on the proposal generation result from BMN, while our single model does not need any extra proposal generation network, suggesting the simplicity of ETAD. (4) At last, we also test different proposal sampling strategies on HACS and find the results are similar to those in ActivityNet. Both random sampling and grid sampling achieve decent performance. Since the block sampling breaks the distribution of different proposals, thus the detection performance is much worse than others.

D. More results on THUMOS dataset

To further verify the effectiveness of proposed gradient sampling in SGS, we also test different gradient sampling ratios on THUMOS dataset under end-to-end training, as shown in Tab. 6. Here, we choose SlowOnly-R50 or TSM-R50 as the feature encoder. In this ablation, since we are using shorter clip (8 frames per clip), lower resolution (180×180), and only RGB modality, the performance is expected to be lower than the state-of-the-art performance listed in Tab.2 of the main paper.

From the results, we can find that if the gradient sampling ratio γ is 0, *i.e.* frozen encoder, the performance is not that promising. However, once we unfreeze the backbone, the performances are instantly boosted with more than 7% gains of average mAP using SlowOnly, and more than 5% gains of average mAP using TSM. This verifies the importance of end-to-end training again. Besides, with different sampling ratios, we interestingly find the performances generally remain at the same level. Such a conclusion is consistent with the ablation results on ActivityNet-1.3 dataset, further proving that a small portion of snippets is enough for end-to-end training in TAD and the effectiveness of our gradient sampling approach.

E. Additional Study of End-To-End Training

In this section, we discuss several factors that are important for the video encoder in end-to-end training, such as data augmentation, frozen backbones, frame resolution, and pretraining. For ablation, we remove the sequential design for all experiments in this section to reflect the real GPU memory consumption.

Data augmentation is vital for end-to-end training. One of the main advantages of end-to-end training is that we can use data augmentation on original frames, which is not possible in feature-based settings. As shown in Tab. 7 (top), we implement random cropping and temporal jittering at the snippet level as data augmentation. Here, temporal jittering means we shift a random stride of each frame in a snippet. Compared with not using data augmentation, random cropping is very helpful for TAD while temporal jittering slightly harms the performance. Therefore, we only use random cropping in our experiment as default.

Partially freeing backbone can have a good trade-off between computation and performance. It is a common trick to save the computation by freezing some shallow layers of video encoder. In this study, we want to know how the frozen layers affect the detection performance. For a ResNet-based encoder (*e.g.* TSM) with four stages, we can gradually freeze the layers from shallow to deep. Tab. 7 (middle) clearly shows that: **1)** End-to-end training is important for TAD, since the frozen stage of 4, *i.e.* freeze the whole backbone, has the lowest mAP compared with others. **2)** As we freeze fewer encoder layers, detection performance will be improved, but the gain becomes smaller, and the memory consumption also becomes much larger. **3)** To have a good trade-off between memory and performance, frozen stage 2 is recommended in our experiments.

Higher frame resolution can boost the performance by a large margin. We also study the impact of the frame resolution on the detection performance in end-to-end training, as shown in Tab. 7 (middle). If we freeze the whole backbone, a higher resolution can boost the mAP from 34.26 to 36.24 (+1.98), which is a significant improvement. If we freeze fewer encoder layers, the gap of mAP between low frame resolution and high frame resolution would be smaller, but still can bring +1.26 gains under frozen stage 2. However, the memory usage is almost doubled in this case.

Classification pretraining is not necessary if using end-to-end training. Some other TAD methods are proposed to finetune the video encoder by the classification task on the target dataset, then extract features [6]. In our case, if we finetune the encoder on ActivityNet by action recognition task, the detection performance would be slightly improved from 36.79 to 36.92, as shown in Tab. 7 (bottom). Such a small gain (+0.13) is reasonable, since end-to-end training can already improve the feature representation from the target domain. Since classification pretraining also takes a long time to train, thus it is not necessary to conduct this pretraining when end-to-end training is adopted.

To summarize, we recommend giving priority to higher frame resolution with stronger data augmentation when with a limited resource budget in end-to-end training. If necessary, we can further freeze certain layers in the encoder to save computation. We believe such a study for end-to-end training will enlighten the TAD community in the sense of efficiency and efficacy trade-off.

Table 7. **Study of different data augmentation, frozen stage, frame resolution, and pretraining of video encoder in end-to-end training** on ActivityNet-1.3. Note that SGS is not adopted in the experiments. We report the GPU memory usage per video with TSM-R50. † means out of memory on a V100 GPU. ‡ means the encoder is finetuned on ActivityNet by the classification task.

Encoder	Frame Resolution	Data Augment.	Frozen Stage	Average mAP	Memory (GB)
TSM	112x112	×	2	35.12	9.1
TSM	112x112	jitter	2	35.17	9.1
TSM	112x112	crop	2	**35.53**	9.1
TSM	112x112	crop+jitter	2	35.38	9.1
TSM	112x112	crop	4	34.26	4.5
TSM	112x112	crop	3	35.01	4.6
TSM	112x112	crop	2	**35.53**	9.1
TSM	112x112	crop	1	35.52	17.0
TSM	112x112	crop	0	35.46	25.8
TSM	224x224	crop	4	36.24	17.5
TSM	224x224	crop	3	36.47	17.6
TSM	224x224	crop	2	**36.79**	34.3
TSM	224x224	crop	1	-	OOM†
TSM-FT‡	224x224	crop	2	**36.92**	34.3

F. Implementation details of SGS

In this section, we describe the implementation details of Sequential Gradient Sampling (SGS). As shown in Alg. 1, SGS can be divided into three stages. First, in sequentialized video encoding, the video is chunked (in temporal dimension) into multiple micro-batches, and feature of each micro-batch is sequentially extracted by the video encoder. Then, the action detector takes the concatenated features, and further, the parameters are updated by the loss. We collect the feature gradients and free all the cache in GPU memory. Last, we sample a portion of feature gradients to save the computation (*e.g.* random sampling), and backward the video encoder by a micro-batch. After sequentially backward all the sampled micro-batches, we accumulate all the gradients and update the video encoders' parameter.

G. Details of Feature-guided Sampling

In the paper, we explore two feature-guided sampling strategies, *i.e.* *farthest point sampling*, and *determinantal point process*. The motivation of feature-guided sampling

Algorithm 1 PyTorch-like Pseudocode of Sequentialized Gradient Sampling.

```
# frames: Nx3xTxHxW, K: micro_batch size
optimizer.zero_grad()

# stage 1: sequentialized video encoding
feats = []
micro_batches = torch.chunk(frames, N//K, dim=0)
for micro_batch in micro_batches:
  with torch.set_grad_enabled(False):
    feat = self.video_encoder(micro_batch)
    feats.append(feat.detach())
feats = torch.stack(feats, dim=0)

# stage 2: action detector learning
feats.requires_grad(True).retain_grad()
with torch.set_grad_enabled(True):
  pred = self.action_detector(feats)
  loss = loss_func(pred, gt)
  loss.backward()
feats_grad = copy.deepcopy(feats.grad.detach())

# stage 3: sequentialized gradient sampling
sample_idx = torch.randperm(N)[:sample_size]
micro_batches = torch.chunk(frames[sample_idx],
    sample_size//K, dim=0)
grads = torch.chunk(feats_grad[sample_idx],
    sample_size//K, dim=0)
for micro_batch, grad in micro_batches, grads:
  with torch.set_grad_enabled(True):
    feat = self.video_encoder(micro_batch)
    feat.backward(gradient=grad)

# update the parameter
optimizer.step()
```

is that each data has different features than others, which means their inherent importance is also different from others. Therefore, one can try to leverage the information inside the data and ensure the most informative or important samples are selected. For TAD, we can adopt feature-guided sampling on the embedding space of samples (*e.g.* snippet features, proposal features). The pseudocode can be found in Alg. 2.

The farthest point sampling (FPS) is a common feature-guided sampling approach, which has been adopted in many fields such as point cloud understanding. Given the data points $X \in \mathbb{R}^{N \times C}$, where N is the number of total samples, and C is the dimension number of each sample feature, FPS selects the new point from the unselected points, and ensures new point has the farthest distance to the currently selected data points in the embedding space. The distance between two different points can be measured by a distance function, which we use euclidean distance in our case. Such sampling actually samples the next point in the middle of the least-known area of the sampling domain, and thus can guarantee the sampled points are most distinguished from each other. However, since this sampling process is conducted iteratively, thus the corresponding time complexity is $O(N^2)$.

Algorithm 2 Pseudocode of FPS/DPP sampling strategy.

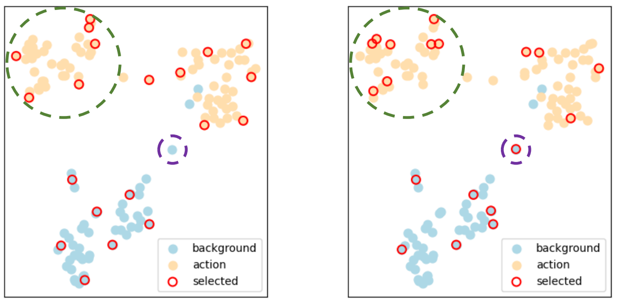
```
# data: N x C, sampling_ratio: (float) 0~1.
# sampling_strategy: fps or dpp.
import torch_cluster
from dppy.finite_dpps import FiniteDPP

N = data.shape[0]
# farthest point sampling
if sampling_strategy == "fps":
  index = torch_cluster.fps(data, ratio=
      sampling_ratio)

# determinantal point process
if sampling_strategy == "dpp":
  data = np.float64(data)
  sample_num = int(sampling_ratio * N)
  # likelihood kernel, use eye matrix to increase
       the rank
  kernel = data.dot(data.T) + 1e-2 * np.eye(N)
  DPP = FiniteDPP("likelihood", **{"L": kernel})
  index = DPP.sample_exact_k_dpp(size=sample_num)

# return the index list of selected samples
```

We also implement another feature-guided sampling as the determinantal point process (DPP). DPP measures the sample probability as a determinant of some kernels. In our case, we use cosine similarity as the kernel function, and update the likelihood matrix every iteration. Since our sampling ratio is fixed, we can use kDPP [4] to approximate DPP for fast sampling. To meet the requirements of kDPP, we add an eye matrix filled with small values (*e.g.* 1e-2) to ensure the rank of the likelihood matrix is larger than the sample size. In general, DPP sampling improves the diversity of sampled data in the embedding space.



(a) FPS (b) DPP

Figure 2. **t-SNE visualization of FPS sampling and DPP sampling**. The orange dots are snippets inside action and the blue dots are background snippets. Dots with red outlines are the sampled snippets.

In our experiments, we find that DPP works always better than FPS, as shown in Tab.4 in the main paper. To further analyze these two strategies, we provide the t-SNE visualization [7] of FPS and DPP at the snippet level, as shown in Fig. 2. In this figure, the yellow points are the action foreground snippets, and the blue points are the background snippets. Initially, we can find that these snippets can be well grouped in different clusters based on their features, which verifies the necessity of conducting sampling on feature embedding space. We observe that **(1)** For points in the dashed green circle, FPS tends to select extreme points, while DPP can select samples with a larger variety. **(2)** For the point in the purple dash circle, FPS misses such a hard-negative sample because its distance from other points is not that big in the embedding space. However, such points may be informative samples, and DPP successfully selects this representative sample. Those two findings can explain the success of DPP for snippet-level gradient sampling.

For DPP and FPS on the proposal level, we notice that FPS would prefer to focus on small-scale proposals since these proposal features are more distinguished from each other in the feature space. Due to a lack of enough learning of middle and large-scale proposals, FPS behaves badly in proposal sampling. On the contrary, DPP can sample all scale proposals and succeed in this case.

H. Qualitative Visualization

In order to provide a more vivid understanding of our method, we visualize the qualitative predictions of our method and BMN [5] on ActivityNet for comparison. In Fig. 3, we plot the ground truth actions of each video (drawn in black and above the black line), and also the top-20 predicted proposals by algorithms (drawn in colors and under the black line). The color of the proposal represents the maximum IoU of this proposal to the ground truth actions. Therefore, a proposal with lighter color means it has more overlap with the ground truth, indicating this is a high-quality proposal.

As demonstrated in the figure, ETAD can generate **(1) more precise proposal boundary.** For instance, in the first and third row in Fig. 3, the boundary of proposals from ETAD is closer to the real action boundary than BMN. **(2) more reliable proposal confidence.** As shown in the first and second row in Fig. 3, ETAD has fewer false positive proposals and proves that regressed proposal confidence is much more reliable than BMN, indicating the advantage of our method on proposal ranking.

References

[1] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6154–6162, 2018. 1

[2] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. SlowFast networks for video recognition. In *ICCV*, 2019. 3

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997. 1

[4] Alex Kulesza and Ben Taskar. k-dpps: Fixed-size determinantal point processes. In *ICML*, 2011. 5

[5] Tianwei Lin, Xiao Liu, Xin Li, Errui Ding, and Shilei Wen. BMN: boundary-matching network for temporal action proposal generation. In *ICCV*, 2019. 2, 3, 6

[6] Zhiwu Qing, Haisheng Su, Weihao Gan, Dongliang Wang, Wei Wu, Xiang Wang, Yu Qiao, Junjie Yan, Changxin Gao, and Nong Sang. Temporal context aggregation network for temporal action proposal refinement. In *CVPR*, 2021. 1, 3, 4

[7] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008. 6

[8] Mengmeng Xu, Chen Zhao, David S Rojas, Ali Thabet, and Bernard Ghanem. G-TAD: Sub-graph localization for temporal action detection. In *CVPR*, 2020. 1

[9] Hang Zhao, Zhicheng Yan, Lorenzo Torresani, and Antonio Torralba. HACS: Human action clips and segments dataset for recognition and temporal localization. *ICCV*, 2019. 3

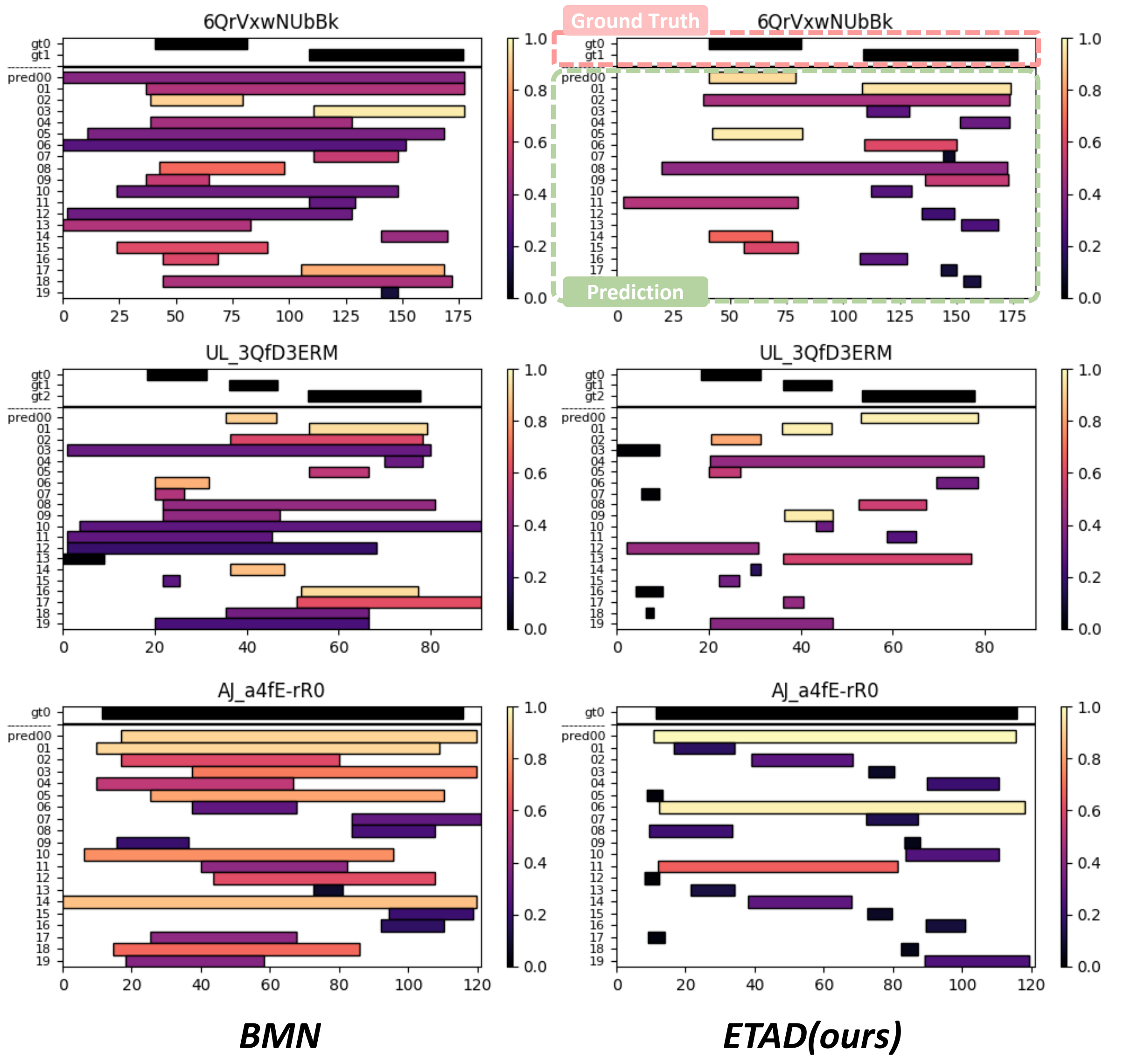


Figure 3. Qualitative results of ETAD and BMN on ActivityNet-1.3. The color of the proposal represents the maximum IoU of this proposal to ground truth actions. We plot the ground truth actions of each video (drawn in black and above the black line), and top-20 predicted proposals by algorithms (drawn in colors and under the black line).