# ETAD: A SAMPLING-BASED APPROACH FOR EFFICIENT TEMPORAL ACTION DETECTION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Temporal action detection (TAD) often suffers from the pain of huge demand for computing resources due to long video duration. As a consequence, given limited resources, most action detectors can only operate on pre-extracted features rather than original video frames, resulting in sub-optimal solutions. In this work, we propose an efficient temporal action detector (ETAD) that can train directly from video frames, by introducing *a novel sampling mechanism*. First, for *where to sample* in TAD, we propose snippet-level sampling and proposal-level sampling, based on the observation that performance saturates at a small number of snippets/proposals. Such samplings essentially leverage the redundancy in the current detection framework, thus can substantially reduce the computation cost and enable end-to-end training for long untrimmed videos without harming the performance. Second, for *how to sample* in TAD, we comprehensively study various sampling approaches, and point out that the random sampling and DPP sampling work the best empirically. Our sampling-based ETAD achieves state-of-the-art performance on TAD benchmarks with remarkable efficiency. With end-to-end training, ETAD can reach **38.25% average mAP** on ActivityNet-1.3. With pre-extracted features, ETAD only needs **6 mins of training time** and **1.23 GB memory**, still reaching average mAP **37.78%**. Code will be available.

## 1 INTRODUCTION

Due to the popularization of mobile devices and social media, video content has been rapidly growing in recent years. Thus, the automatic video understanding is in high demand in both academia and industry (Feichtenhofer et al., 2016; Carreira & Zisserman, 2017). Temporal action detection, referred to as TAD, is a fundamental but challenging task in this area, which requires an intelligent system to predict the start and end time of all activity instances in a video, and to identify their categories as well. (Heilbron et al., 2015; Escorcia et al., 2016; Zhao et al., 2019; Xu et al., 2020)

Most of the current TAD solutions follow the two-step framework: *feature extraction + action detection*. Concretely, a video understanding model first encodes the video as a sequence of feature vectors, then an action detector operates on the features to produce predictions. The well-developed resource-demanding video understanding model and action detection model lead to two efficiency issues: (1) Infeasible to end-to-end training. Ideally, optimizing the feature encoder and action detector jointly is welcomed to avoid the sub-optimal issue. However, such an approach is extremely challenging in TAD due to the large data volume of untrimmed videos under limited GPU storage. As a result, most existing works in TAD either only focus on learning a good action detector with the pre-encoded video features (Bai et al., 2020; Xu et al., 2020; Zhao et al., 2021), or explore end-to-end training by compromising the data resolution or batch size to satisfy the memory requirements, which may influence the network performance (Xu et al., 2017; Lin et al., 2021). (2) Even with pre-encoded features, a standalone action detector itself is usually not resource-friendly, especially when designed with expensive modules in order to pursue good detection performance. For example, many recent solutions incorporate the graph convolutions (Xu et al., 2020; Zhao et al., 2021), attentions (Sridhar et al., 2021), or transformers (Zhang et al., 2022), which all incur high computation and memory costs by correlating different snippets or proposals exhaustively. Therefore, an efficient action detection model becomes increasingly important for TAD.

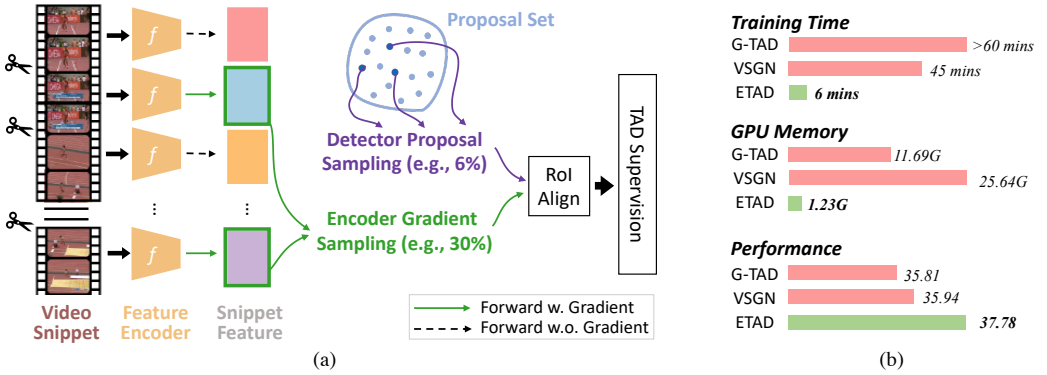



Figure 1: **Overview of ETAD.** (a) By applying Encoder Gradient Sampling (green arrows) and Detector Proposal Sampling (purple arrows), ETAD is able to optimize the feature encoder and action detector jointly with low GPU memory footprint. We find that only 30% snippets and 6% proposals are sufficient for TAD. (b) With pre-extracted features, ETAD can outperform other state-of-the-art methods with much less training time and GPU memory (tested on a single RTX3090).

In this paper, to alleviate the aforementioned heavy computation issue in TAD, we propose a novel sampling-based end-to-end Efficient Temporal Action Detector (ETAD), as shown in Figure 1(a). The success of ETAD on efficacy and efficiency is achieved by answering two key questions, *i.e. where to sample* and *how to sample* in TAD. Firstly, for *where to sample* in the network, we design two sampling processes: snippet-level sampling and proposal-level sampling. The gradient-based snippet-level sampling, named **Encoder Gradient Sampling (EGS)**, reduces the number of redundant snippets for gradient computation during network back-propagation, which not only significantly alleviates the GPU memory and computation cost incurred by the video encoder, but also enables the detector working with deep architecture, high frame resolution, or large batch size during end-to-end training. The proposal-level sampling, named **Detector Proposal Sampling (DPS)**, generates a much smaller but sufficient set of proposals for the detector to further process, based on the empirical observation that most of the candidate proposals overlap with each other and only 6% proposals can guarantee a decent action detection performance. Second, for *how to sample*, we explored various sampling strategies at both snippet and proposal levels, including heuristic sampling strategies (*e.g.* random sampling), label-guided sampling strategies (*e.g.* IoU-balanced sampling), and feature-guided sampling strategies (*e.g.* farthest point sampling, determinantal point process (DPP)). The experimental results suggest that random sampling and DPP sampling advance in simplicity and performance at these two levels, respectively.

ETAD achieves state-of-the-art performance on ActivityNet-1.3 and THUMOS benchmarks with superior performance and efficiency. Interestingly, we find that with our effective sampling strategies, it is sufficient to have a promising action detector while only leveraging 30% of snippets for back-propagation and 6% of proposals for detection. Compared with the state-of-the-art method VSGN (Zhao et al., 2021), Figure 1(b) shows that ETAD only requires 5% of VSGN's memory usage (1.23GB vs 25.64GB) and 13% of training time (6mins vs 45mins), and it improves the average mAP from 35.94% to 37.78% on ActivityNet-1.3.

**Contributions.** (1) We novelly propose to alleviate the efficiency issue in TAD by the sampling mechanism. Concretely, we are the first to explicitly point out that snippet-level sampling and proposal-level sampling can significantly reduce the learning redundancy in TAD. Particularly, the proposed gradient sampling can serve as a powerful approach for efficient end-to-end training for long untrimmed videos. (2) We compare various sampling strategies both at the snippet level and proposal level. Empirically, the random sampling and DPP-based sampling exceed others in performance. **(3)** Extensive experiments and ablations show that our proposed ETAD reaches state-of-the-art performance on two commonly used TAD benchmarks, ActivityNet-1.3 and THUMOS-14.

## 2 Related Work

**Temporal Action Detection.** An action detector can localize action instances directly from features of video snippets (*direct*), or merely refine the action boundaries of proposals from a proposal-generation network (*refinement*). The *direct* solution usually consists of feature enhancement and anchor/proposal evaluation, and focuses on improving one or both components (Heilbron et al., 2017; Xu et al., 2017; Chao et al., 2018; Lin et al., 2019b; Long et al., 2019). For example, Xu et al. (2020) models the correlations between video snippets for feature enhancement, and Bai et al. (2020) models the correlations between proposals for proposal evaluation. The *refinement* solution tends to prune off-the-shelf action proposals and provide more accurate boundary predictions (Shou et al., 2016; Lin et al., 2018; Qing et al., 2021; Zhao et al., 2021), where the proposal can be from either a heuristic design (*e.g.* sliding window), or a well-developed proposal generation method (Escorcia et al., 2016; Buch et al., 2017; Gao et al., 2018; Liu et al., 2019). P-GCN (Zeng et al., 2019) is a typical refinement solution that exploits proposal-proposal relations from the predictions of BSN (Lin et al., 2018) and uses graph convolutional networks to refine proposals. Qing et al. (2021) uses the high-quality proposals generated from BMN (Lin et al., 2019b) and proposes a cascade structure to progressively refine the actions. Our proposed ETAD belongs to the family of direct solutions, since it does not rely on any external proposal methods, but surpasses the best refinement solution.

**End-to-end Solutions in TAD.** Few methods study the problem of action detection directly from the original video frame end to the predicted proposal end without stopping the gradients in the middle of the pipeline, which is referred to as *end-to-end training*. Xu et al. (2017) encodes the frames with 3D filters, proposes action segments, then classifies/refines them. However, its performance is limited due to underlying computational restrictions (*e.g.* low-resolution frames and low-capacity video networks). PBRNet (Liu & Wang, 2020) progressively refines action boundaries with three cascaded detection modules, but it is also plagued by hardware constraints that necessitate low-resolution frame input. ASFD (Lin et al., 2021) proposes the first anchor-free temporal localization method, but it suffers from a low convergence rate and small batch size. To approximately overcome this issue, some works propose ways of pre-training the feature encoder. Those methods develop new training tasks to close the gap between action recognition and action detection. For example, Xu et al. (2021b) reduces the mini-batch composition in terms of temporal, spatial or spatio-temporal resolution. Xu et al. (2021a) and Alwassel et al. (2021) train the feature encoder on novel tasks to make video features sensitive to the temporal localization of the action. Unfortunately, they both require an extra pretraining stage for training the localization network. Differently, our ETAD solution is able to train the network with full fidelity: high frame resolution, large batch size, fast convergence, and single-stage training.

**Sampling in Video Understanding.** Although densely sampling snippets over the entire sequence is effective in understanding short videos, it is expensive for long, untrimmed videos. An alternative approach is to summarize the video (Huang & Wang, 2019) by selecting only the relevant frames or snippets. In action recognition, for example, SCSampler (Korbar et al., 2019) selects salient clips from video for efficient action recognition, but it has to train another model as clip sampler. In action detection, sampling is less explored. Cheng & Bertasius (2022) stochastically updates snippet features stored in a pre-computed feature bank, but the unselected intermediate features are not always up-to-date. Besides, proposal sampling in TAD is an under-explored topic because most methods (Lin et al., 2020; 2019b; Xu et al., 2020) exhaustively enumerate the possible locations of activity, leading to redundant computation, especially for highly overlapped proposals. We extensively studied snippet sampling and proposal sampling in TAD, and our ETAD does not need any extra model/storage to select/cache samples.

## 3 Method

Our method is designed to answer two questions in TAD: *where to sample* and *how to sample*. We will firstly introduce the basic architecture of our ETAD, then explain our two sampling processes at the snippet level and proposal level, and finally discuss the possible sampling approaches, including heuristic sampling, feature-guided sampling, and label-guided sampling.

### 3.1 Problem Definition and Model Architecture

Given an untrimmed video, temporal action detection aims to predict actions with temporal boundaries denoted as $\Psi = \{\varphi_i = (t_s, t_e, c)\}_{i=1}^{N}$, where $(t_s, t_e, c)$ are the start time, end time, and category

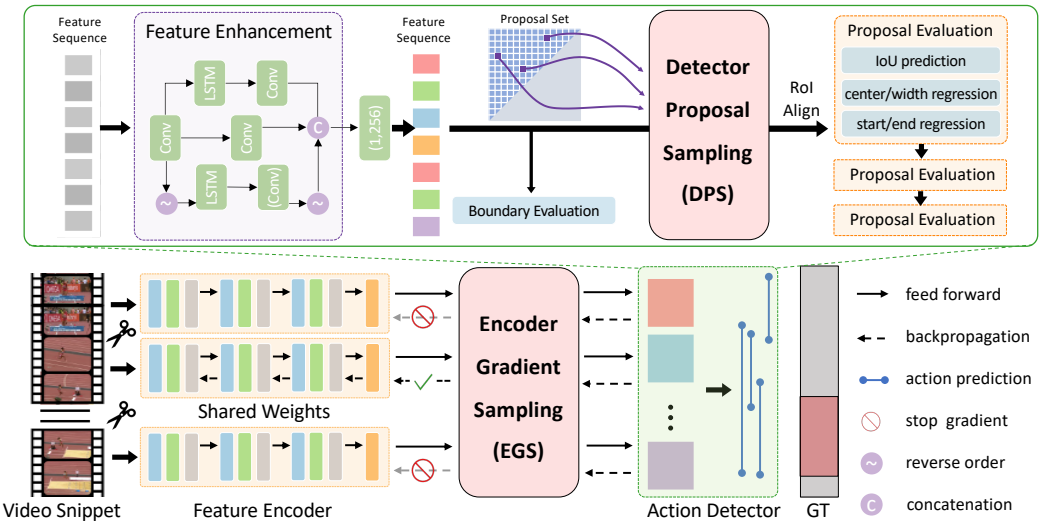



Figure 2: **Architecture of ETAD.** *Bottom*: EGS connects the two components of ETAD — feature encoder and action detector. It enables partial gradients back-propagation at the snippet level, tremendously reducing memory and computational cost. *Top*: DPS, in the action detector of ETAD, samples a small portion at the proposal level for further proposal evaluation.

of the action instance $\varphi_i$, respectively. $N$ is the total number of actions. The overall architecture of our ETAD is shown in Figure 2(bottom), which follows the pipeline of feature extraction and action detection. For feature extraction, an off-the-shelf action recognition model, such as TSM (Lin et al., 2019a), R(2+1)D (Tran et al., 2018), is adapted to encode the video frames to a list of feature vectors. Specifically, each feature vector is from the feature map before the classification head of the recognition model, and with global average pooling applied on the temporal and spatial dimensions.

The architecture of our action detector is shown in Figure 2(top). In the feature enhancement module, we integrate two LSTM layers to capture long-range temporal relations, enhancing snippet-level representations. In the boundary evaluation, we apply two convolution layers to classify the startness and endness of each snippet. Last, the proposal evaluation module refines the candidate anchors/proposals and outputs their confidence. In this module, we apply temporal RoI alignment from G-TAD (Xu et al., 2020) for proposal feature extraction. To improve the regression precision, we stack more proposal evaluation modules with progressively improved IoU thresholds. More details about each module can be found in Appendix A.

The architecture with the aforementioned modules without the sampling mechanism is called our vanilla model, which can produce decent-quality action predictions. Although more sophisticated modules, such as self-attention or graph convolution networks, can be integrated into our action detector to achieve higher performance, we show that a simple detector can be more computationally efficient, even with end-to-end training, without losing much performance. To be more specific, we propose to alleviate the computation burden by adopting ***the sampling mechanism***. We detailed investigate this mechanism in two questions: *where to sample* and *how to sample*.

### 3.2 Where to sample in TAD

The heavy computation burden in many SOTA TAD methods, including our vanilla model, generally comes from two places: snippet-level redundancy, and proposal-level redundancy.

#### 3.2.1 Snippet-level sampling

Ideally, for end-to-end training in TAD, the input of feature encoder is all snippets in a sequence (a sequence is part of a video where a TAD model processes at one time) and the feature encoder's parameters are optimized based on all the input snippets' gradients in each iteration. However, such full end-to-end optimization is usually not feasible in practice due to the conflict between the large

Table 1: Comparison of ETAD with other *(approximate) end-to-end training* strategies in TAD. For a fair comparison, we set the sampling rate to 30% in all experiments, and use the vanilla ETAD detector with different end-to-end strategies. Computation in forward/backward stands for the theoretical computation cost of the feature encoder in forward/backward propagation. GPU memory per video is reported with TSM-ResNet50 backbone.

| Methods | Computation in Forward | Computation in Backward | Feature Bank | GPU Memory | avg. mAP |
|---|---|---|---|---|---|
| Pre-extracted Feature | 0% | 0% | ✓ | 1.2GB | 36.13 |
| Multi-stage Training (Xu et al., 2020) | 30% + 30% | 30% + 30% | ✗ | 11GB | 36.36 |
| Feature Bank (Cheng & Bertasius, 2022) | 30% | 30% | ✓ | 11GB | 36.54 |
| **Gradient Sampling (ETAD, ours)** | **100%** | **30%** | ✗ | **11GB** | **36.79** |
| Completely End-to-End Training | 100% | 100% | ✗ | 34GB | 36.85 |

data volume and the GPU memory constraint. For example, when using a large number of snippets (*e.g.* 128 snippets), a high spatial resolution (*e.g.* $224 \times 224$), a large batch size (*e.g.* 16 videos), and a mediate-capacity encoder (*e.g.* TSM), more than 500 GB memory is needed to perform end-to-end training, making it unrealistic on most platforms. Some works try to sacrifice the input resolution or network scale to reduce the end-to-end computation, but these techniques may degrade the TAD performance (Liu & Wang, 2020; Liu et al., 2022; Yang et al., 2022). On the contrary, we target approaches that avoid sacrificing the data or network scale.

In this paper, we propose to reduce the gradient computation by a *snippet-level sampling* strategy, called **Encoder Gradient Sampling (EGS)**, as illustrated in Figure 2 (bottom). Given a sampling ratio, we sample snippets using a sampling strategy (Section 3.3), and only compute the encoder's gradients on the sampled snippets in backward propagation of the encoder. The rest of snippets only contribute to the forward computation. This gradient-based sampling saves a large portion of the GPU memory, since for unsampled snippets intermediate activations do not need to be saved to compute gradients. Thus it also allows the network to accommodate frames of higher resolutions, or an encoder of larger capacity.

EGS makes end-to-end TAD more efficient, and why can it keep its effectiveness? (1) Since the consecutive video frames in the untrimmed video are usually similar in semantics, the feature vectors of corresponding snippets may share similar representations. When end-to-end training is adopted, the encoder gradients of such snippets behave quite similarly, suggesting the learning redundancy. (2) As pointed out by Cheng & Bertasius (2022), the off-the-shelf feature encoder is already pre-trained on a large-scale action recognition dataset, thus the encoder evolves more slowly than the other modules in the network with a much smaller learning rate. Therefore, the difference in the feature encoder's gradient between snippets is even more smaller. Based on such insight, our EGS which only back-propagates a small ratio of snippets would still guarantee high TAD performance.

We also compare our method with other *(approximate) end-to-end training* approaches in TAD, as shown in Table 1. To meet the requirements of memory, LoFi (Xu et al., 2020) introduces three stages for encoder pretraining, which downsamples the temporal/spatial resolution, then extracts features and detects actions in full fidelity. TALLFormer (Cheng & Bertasius, 2022) designs a feature bank to store snippet features as cache, and stochastically updates only a small proportion of these to approximate end-to-end training, but the features in the bank are not always up to date. As a comparison, our EGS shares a similar GPU memory with these two methods because of the computation reduction in backward propagation, but it achieves the best detection performance (see Table 1). Noted that our EGS is agnostic of the encoder architecture and thus can incorporate any of the common encoders in its framework. The detailed implementation of EGS can be found in the Appendix G.

#### 3.2.2 PROPOSAL-LEVEL SAMPLING

Beyond the snippet-level sampling that is only designed for end-to-end training, in this paper, we also propose the *proposal-level sampling*, which aims to reduce the redundancy in the dense proposal evaluation in action detector. In current two-stage TAD methods (*i.e.* methods use RoI alignment or similar to extract proposal features explicitly), a dense candidate proposal set is needed in the

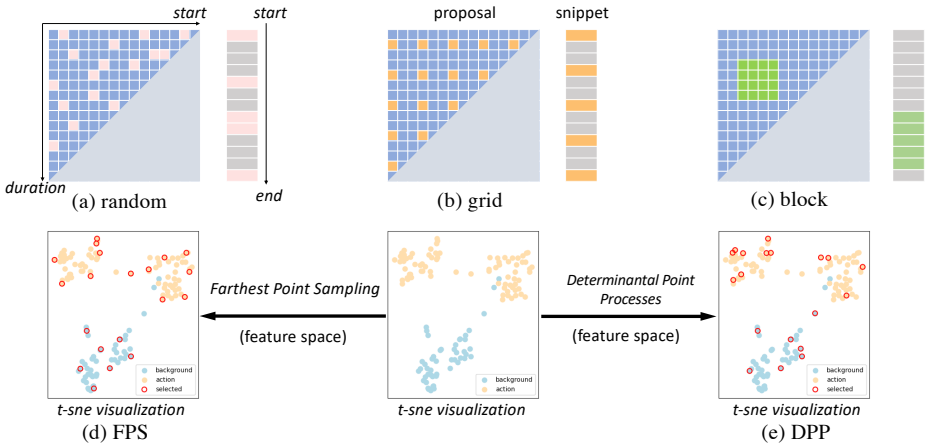



Figure 3: **Different sampling strategies: *random*, *grid*, *block*, *FPS*, *DPP*.** *Random* strategy samples certain proposals/snippets from a uniform distribution, *grid* strategy samples with a fixed temporal stride, and *block* strategy samples a consecutive area in proposal map/snippet sequence. Besides, *FPS* chooses the new snippet/proposal which has the farthest distance to the selected samples. *DPP* selects the data from the feature embedding space to enforce the sample diversity.

second stage for proposal refinement and further processing, such as in BMN (Lin et al., 2019b) and G-TAD (Xu et al., 2020). To deal with the large action length variation, they propose to enumerate all possible combinations of start and end locations to form candidate proposals. Specifically, given the number of snippets $T$, there will be $C_T^2 = T \cdot (T-1)/2$ proposals, which has the quadratic complexity with respective to $T$.

However, due to the dense enumeration, most of these proposals overlap with each other, and as a result, a large portion of the extracted proposal features are duplicated. Moreover, the proposal evaluation module also refines each proposal's start and end boundary, making it unnecessary to consider proposals that are temporally close. Therefore, huge redundant computation exists in the current proposal extraction and evaluation modules. In order to reduce such redundancy while preserving performance, we propose to replace the densely sampled proposal set with a subset produced by an efficient sampling, called **Detector Proposal Sampling (DPS)**, as illustrated in Figure 2. Our experiment results suggest that with a decent sampling strategy, only 6% proposals can provide a similar detection performance to all proposals, but saves more than 90% computation.

Note that though DPS can be applied on both the pre-extracted feature setting and the end-to-end training setting, the relative efficiency gains brought by this technique are more obvious with pre-extracted features. This is because in end-to-end training, the action detector where DPS is applied uses much lower computation and memory than the feature encoder, and the savings by DPS is not that significant in the whole pipeline. However, considering that most of the current TAD methods are feature based, this sampling strategy is important to make this type of TAD methods even more efficient. For example, DPS can speed up the training time of ETAD from 45 mins to 6 mins in feature-based setting, requiring only 1.2GB GPU memory.

### 3.3 How to sample in TAD

In previous sections, both EGS and DPS require to determine a subset of candidate snippets / proposals, thus they can share similar sampling strategies — criteria to choose the samples. In our study, we explore three types of sampling strategies: *heuristic sampling*, *feature-guided sampling*, and *label-guided sampling*.

**Heuristic Sampling** includes three simple strategies: random, grid, and block, which are similar to the samplings in MAE (He et al., 2022). Given a sampling ratio, the *random* strategy simply samples snippets or proposals randomly following a uniform sampling distribution. The *grid* strategy samples the snippets with a pre-defined temporal stride or grid on a proposal map, as shown in Figure 3. The *block* sampling strategy makes sure consecutive snippets are sampled, or the proposals within a block in the proposal map are sampled. The block sampling essentially evaluates the model in a trimmed clip of the video. The sampling process of such trio does not rely on other conditions, such as proposal/snippet feature, thus we call them heuristic sampling.

**Feature-guided Sampling**, which are more sophisticated sampling strategies, are based on the distribution in the feature space. The farthest point sampling (FPS, Eldar et al. (1997)) selects the new snippet/proposal which has the farthest distance to the selected samples. Here, the distance between two snippets or proposals is defined as the euclidean distance between two snippet features or proposal features. Such a method can provide the most distinguished samples of the candidates, since the selected samples are more variant in the embedding space. We also implement the sampling as the determinantal point process (DPP) to enforce diversity during training. We take the cosine similarity as the kernel function, and update the determinant every training epoch. When a sampling ratio is given, the sample target has a fixed size, so we can directly apply kDPP (Kulesza & Taskar, 2011) in the scenario. Please refer to the Appendix H for more details.

**Label-guided Sampling** is a category that uses groundtruth supervision. For proposal-level sampling, we explore the following strategies. The IoU-balanced sampling guarantees the selected proposals has nearly the same number in different IoU threshold. For instance, we make sure the proposal number is the same based on their IoU around 0∼0.3, 0.3∼0.7, and 0.7∼1. Similarly, the scale-balanced sampling tries to maintain the equivalence of proposal numbers around different action scales, meaning the number of small scale proposals (scale$<$0.3), middle scale proposals (0.3$<$scale$<$0.7), and large scale proposals (scale$>$0.7) are the same in the selected sample set.

In our experiment, we find that random sampling, grid sampling, and DPP-based sampling all work well in terms of performance. Using a rather small sampling rate, *e.g.* 30% at the snippet level and 6% at the proposal level, can produce a decent TAD performance. Such small sampling ratios can save a lot of computation both in feature encoder and action detector while maintaining the performance.

# 4 Experiments

## 4.1 Implementation Details

**Datasets and evaluation metrics.** ActivityNet-1.3 (Heilbron et al., 2015) is a large-scale video understanding dataset, consisting of 19,994 videos annotated for the temporal action detection task. The dataset is divided into train, validation and test sets with a ratio of 2:1:1. THUMOS-14 (Jiang et al., 2014) contains 200 annotated untrimmed videos in the validation set and 213 videos in the test set. Following previous works, mean Average Precision (mAP) at certain IoU thresholds and average mAP are reported as the main evaluation metrics. On ActivityNet-1.3, the IoU thresholds are chosen from 0.5 to 0.95 with 10 steps. On THUMOS-14, the threshold is chosen from $\{0.3,0.4,0.5,0.6,0.7\}$.

**Feature preparation.** In this paper, we study two settings: pre-extracted feature setting, and end-to-end training setting. TSN (Wang et al., 2016), TSM (Lin et al., 2019a), I3D (Carreira & Zisserman, 2017), and TSP (Alwassel et al., 2021) are adopted as the feature encoder in different settings. We implement our framework with PyTorch1.12, CUDA11.1, and mmaction2 (Contributors, 2020). The feature-based experiments are conducted on one RTX3090, and the end-to-end experiments are conducted on 8 A100 GPUs. Please refer to Appendix B for more implementation details and Appendix C for more ablations. For completeness, we also evaluate our methods on the HACS dataset (Zhao et al., 2019) and achieve state-of-the-art performance (see Appendix D).

## 4.2 Comparison with State-of-the-Art TAD Methods

**Pre-extracted feature setting.** We first use the pre-extracted features to compare our action detector with other state-of-the-art methods on ActivityNet-1.3. This provides an apple-to-apple comparison with other TAD methods which are not able to optimize the video encoder jointly. All results are the average after training 5 times. As shown in Table 2, our method achieves the highest average mAP on this large-scale dataset. Notably, without expensive optical flow, our localization network can reach an average mAP of 37.78%, significantly outperforming other methods. We also show the advantage of our method on THUMOS-14 in Table 3, which surpasses both TSN-based VSGN (Zhao et al., 2021) and I3D-based AFSD (Lin et al., 2021) and MUSES (Liu et al., 2021). Interestingly, ETAD achieves outstanding results especially on high IoU threshold on the two datasets, indicating the precision of the generated action boundary.

**End-to-end training setting.** Table 4 compares ETAD with other detection methods that adopt end-to-end training. Compared with LoFi (Xu et al., 2021b) which also uses TSM-ResNet50 as the

Table 2: Comparison in terms of mAP on ActivityNet-1.3 validation set **with pre-extracted features**. The O.F. stands for optical flow.

| Method | Feat. | O.F. | 0.5 | 0.75 | 0.95 | Avg. |
|---|---|---|---|---|---|---|
| BMN | TSN | ✓ | 50.07 | 34.78 | 8.29 | 33.85 |
| VSGN | TSN | ✓ | 52.38 | 36.01 | 8.37 | 35.07 |
| TCANet | TSN | ✓ | 52.27 | 36.73 | 6.86 | 35.52 |
| RCL | TSN | ✓ | **54.19** | 36.19 | 9.17 | 35.98 |
| **ETAD** | TSN | ✓ | 52.49 | **37.07** | **9.47** | **36.06** |
| G-TAD | TSP | ✗ | 51.26 | 37.12 | 9.29 | 35.81 |
| CSA | TSP | ✗ | 52.64 | 37.75 | 7.94 | 36.25 |
| RCL | TSP | ✗ | **55.15** | 39.02 | 8.27 | 37.65 |
| **ETAD** | TSP | ✗ | 54.70 | **39.10** | **9.87** | **37.78** |

Table 3: Comparison in terms of mAP on THUMOS-14 test set **with pre-extracted features**. Optical flow is used in all the methods.

| Method | Feat. | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 |
|---|---|---|---|---|---|---|
| BMN | TSN | 56.0 | 47.4 | 38.8 | 29.7 | 20.5 |
| G-TAD | TSN | 57.3 | 51.3 | 43.0 | 32.6 | 22.8 |
| TCANet | TSN | 60.6 | 53.2 | 44.6 | 36.8 | 26.7 |
| VSGN | TSN | 66.7 | 60.4 | 52.4 | 41.0 | 30.4 |
| **ETAD** | TSN | **68.26** | **63.63** | **55.65** | **45.83** | **35.13** |
| AFSD | I3D | 67.3 | 62.4 | 55.5 | 43.7 | 31.1 |
| MUSES | I3D | 68.9 | 64.0 | 56.9 | 46.3 | 31.0 |
| TAGS | I3D | 68.6 | 63.8 | 57.0 | 46.3 | 31.8 |
| **ETAD** | I3D | **70.29** | **65.26** | **57.81** | **47.88** | **36.25** |

Table 4: Comparison in terms of mAP on ActivityNet-1.3 **with end-to-end training**. We report per video GPU memory usage. $\gamma$ is encoder gradient sampling ratio. † means additional optical flow is used for late fusion.

| Method | Encoder Backbone | 0.5 | 0.75 | 0.95 | Avg. mAP | Memory (GB) |
|---|---|---|---|---|---|---|
| AFSD † | I3D | 52.40 | 35.30 | 6.50 | 34.40 | 12 |
| LoFi | TSM-ResNet50 | 50.91 | 35.86 | 8.79 | 34.96 | 29 |
| TALLFormer | Swin-B | 54.10 | 36.20 | 7.90 | 35.60 | 29 |
| **ETAD** ($\gamma = 30\%$) | TSM-ResNet50 | 53.79 | 37.59 | 10.56 | 36.79 | 11 |
| **ETAD** ($\gamma = 30\%$) | R(2+1)D-34 (TSP) | **55.49** | **39.32** | **10.57** | **38.25** | **5.5** |

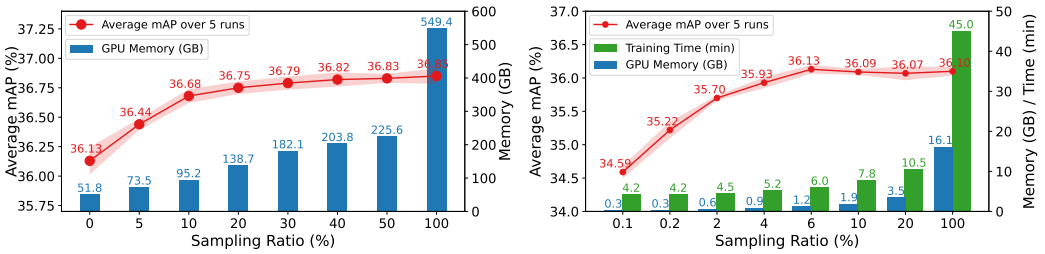
encoder, ETAD can achieve 1.83 average mAP gains with only 38% GPU budget. Particularly, using our EGS method with the R(2+1)D encoder, we can achieve an even higher performance of 38.25% average mAP with only 5.5 GB memory (per video), thus outperforming previous state-of-the-art methods both on efficiency and efficacy by a large margin. Finally, compared with pre-extracted feature experiments above, joint optimization of feature encoder and action detector can boost the performance from 37.78 (in Table 2) to 38.25, suggesting the significance of end-to-end training.

**Qualitative Visualization.** The visualization in Appendix I shows that ETAD can generate more precise proposal boundary and more reliable proposal confidence compared with other methods.

### 4.3 Ablation Study

**Effect of Sampling Ratio**. We first study the impact of different sampling ratios on the action detection performance. **(1)** For snippet-level sampling, as shown in Figure 4(a), if sampling ratio is 100%, *i.e.* the network is trained in completely end-to-end way, it takes tremendous 550 GB memory to optimize (batch size 16), bringing a 0.72 mAP gain compared with sampling ratio 0% (*i.e.* feature-based experiment). However, completely end-to-end training is not necessary. For example, only 10% of the samples contribute to 80% of the improvement, indicating that certain redundancy exists in snippet-level end-to-end learning. Such scenario also happens in THUMOS dataset (see Appendix E). Thus as a good trade-off between performance and efficiency, we take 30% EGS ratio as the main setting, which provides 92% (0.66) mAP gains but reduces GPU memory usage by 66%. This reduction makes end-to-end training of long-form video understanding practical on most platforms, *e.g.* four V100 GPUs for training, single RTX3090 GPU for inference. **(2)** For proposal-level sampling, as shown in Figure 4(b), the detection performance also saturates from a small sampling ratio. Using 6% sampling, ETAD successfully maintains the same detection performance as using complete dense proposal set, speeds up the training 7.5x faster (from 45 mins to 6 mins), and cuts down the 92% memory usage (from 16 G to 1.2 G) of the action detector. Therefore, we can conclude that the training redundancy exists at proposal level and snippet level, and we can apply sampling in both processes to improve efficiency.

**Effect of Sampling Strategy**. We further study different sampling strategies on ActivityNet-1.3, as shown in Table 5. **(1)** For proposal-level sampling (*i.e.* DPS), we find that random sampling, grid sampling, and DPP works all well. While block sampling and label-guided sampling both show certain downgrades in performance, because they change the proposal distribution and thus can not guarantee the variety of proposals. We also 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 this




(a) Encoder Gradient Sampling (b) Detector Proposal Sampling

Figure 4: **Effect of sampling ratio.** (a) For end-to-end training, we apply TSM-R50 with different EGS ratios and report the mAP and GPU memory on ActivityNet-1.3. (b) For feature-based training, we use TSM features with different DPS ratios and report the mAP, GPU memory, and training time. Random sampling is adopted in both settings. Note that the $x$-axis is not uniformly distributed.

Table 5: **Effect of sampling strategies.** We apply the TSM-R50 as the feature encoder and report the mAP on ActivityNet. Since EGS adopts end-to-end training, thus the performance of EGS is expected to be higher than DPS which uses pre-extracted features.

| Sampling Type | Sampling Strategy | DPS (feauture, 6%) | EGS (end-to-end, 30%) |
|---|---|---|---|
| *heuristic* | random | **36.13** | 36.65 |
| | grid | 36.04 | **36.77** |
| | block | 32.97 | 36.74 |
| *feature-guided* | farthest point sampling (FPS) | 33.59 | 36.61 |
| | determinantal point process (DPP) | **36.16** | **36.79** |
| *label-guided* | IoU-balanced | 34.84 | N.A. |
| | Scale-balanced | 35.10 | N.A. |

experiment. **In short, for proposal-level sampling, we find it vital to preserve the original distribution of all proposals.** **(2)** For snippet-level sampling (*i.e.* EGS), all experiments outperform the pre-extracted feature baseline (36.13%), which also validates the effectiveness of end-to-end training. **We find grid sampling and DPP sampling strategy work slightly better than other strategies in EGS.** Besides, we notice that DPP shows the best performance both on DPS and EGS, since it enforces the diversity of proposals/snippets during training in the feature embedding space. However, considering the detection performance and computation complexity of different sampling strategies, we recommend adopting the random sampling strategy at the proposal level, and adopting grid or DPP sampling strategies at the snippet level.

**Comparison with other end-to-end strategies in the literature**. As discussed in Section 3.2.1, we also adopt ETAD detector with different *(approximate) end-to-end* strategies for comparison. As shown in Table 1, our simple but effective EGS evidently improves the detection performance under the same memory budget. The mAP is nearly the same as in completely end-to-end training, but saves more than 65% memory, suggesting the effectiveness of our method. Besides, additional studies of end-to-end training regarding frame resolution, pretraining, etc, are detailed in Appendix F.

## 5 Conclusion

In this paper, we systematically study the efficiency issue in temporal action detection. We identify that computation redundancy is a major bottleneck in TAD, and thus suggest alleviating such problem by adopting *the sampling mechanism*. We explicitly propose to adopt samplings at snippet-level and proposal-level, and we further explore various sampling strategies on both levels. Our proposed ETAD not only exceeds other methods on pre-extracted feature settings, but also enables joint optimization of the feature encoder and action detector on an affordable GPU budget. ETAD achieves state-of-the-art action detection performance on multiple benchmarks, and the proposed sampling methods and strategies make end-to-end training tractable in real-world applications. This work will encourage the community to carry out more research on end-to-end training for various untrimmed video understanding tasks besides TAD, such as video language grounding and video captioning.

## REFERENCES


Humam Alwassel, Silvio Giancola, and Bernard Ghanem. TSP: Temporally-sensitive pretraining of video encoders for localization tasks. In *ICCV workshops*, 2021.

Yueran Bai, Yingying Wang, Yunhai Tong, Yang Yang, Qiyue Liu, and Junhui Liu. Boundary content graph neural network for temporal action proposal generation. In *ECCV*, 2020.

Shyamal Buch, Victor Escorcia, Chuanqi Shen, Bernard Ghanem, and Juan Carlos Niebles. Sst: Single-stream temporal action proposals. In *CVPR*, 2017.

Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the Kinetics dataset. In *CVPR*, 2017.

Yu-Wei Chao, Sudheendra Vijayanarasimhan, Bryan Seybold, David A. Ross, Jia Deng, and Rahul Sukthankar. Rethinking the faster R-CNN architecture for temporal action localization. In *CVPR*, 2018.

Feng Cheng and Gedas Bertasius. Tallformer: Temporal action localization with long-memory transformer. In *ECCV*, 2022.

MMAction2 Contributors. Openmmlab's next generation video understanding toolbox and benchmark. github.com/open-mmlab/mmaction2, 2020.

Yuval Eldar, Michael Lindenbaum, Moshe Porat, and Yehoshua Y Zeevi. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing*, 6(9):1305–1315, 1997.

Victor Escorcia, Fabian Caba Heilbron, Juan Carlos Niebles, and Bernard Ghanem. DAPs: Deep action proposals for action understanding. In *ECCV*, 2016.

Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *CVPR*, 2016.

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

Jiyang Gao, Kan Chen, and Ramakant Nevatia. CTAP: Complementary temporal action proposal generation. In *ECCV*, 2018.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, 2022.

Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. ActivityNet: A large-scale video benchmark for human activity understanding. In *CVPR*, 2015.

Fabian Caba Heilbron, Wayner Barrios, Victor Escorcia, and Bernard Ghanem. SCC: Semantic context cascade for efficient action detection. In *CVPR*, 2017.

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

Cheng Huang and Hongmei Wang. A novel key-frames selection framework for comprehensive video summarization. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(2): 577–589, 2019.

YG Jiang, J Liu, A Roshan Zamir, G Toderici, I Laptev, M Shah, and R Sukthankar. Thumos challenge: Action recognition with a large number of classes, 2014.

Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017.

Hassan Keshvari Khojasteh, Hoda Mohammadzade, and Hamid Behroozi. Temporal action localization using gated recurrent units. *The Visual Computer*, 2022.

Bruno Korbar, Du Tran, and Lorenzo Torresani. Scsampler: Sampling salient clips from video for efficient action recognition. In *ICCV*, 2019.

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

Chuming Lin, Jian Li, Yabiao Wang, Ying Tai, Donghao Luo, Zhipeng Cui, Chengjie Wang, Jilin Li, Feiyue Huang, and Rongrong Ji. Fast learning of temporal action proposal via dense boundary generator. In *AAAI*, 2020.

Chuming Lin, Chengming Xu, Donghao Luo, Yabiao Wang, Ying Tai, Chengjie Wang, Jilin Li, Feiyue Huang, and Yanwei Fu. Learning salient boundary feature for anchor-free temporal action localization. In *CVPR*, 2021.

Ji Lin, Chuang Gan, and Song Han. TSM: Temporal shift module for efficient video understanding. In *ICCV*, 2019a.

Tianwei Lin, Xu Zhao, Haisheng Su, Chongjing Wang, and Ming Yang. BSN: Boundary sensitive network for temporal action proposal generation. In *ECCV*, 2018.

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

Qinying Liu and Zilei Wang. Progressive boundary refinement network for temporal action detection. In *AAAI*, 2020.

Xiaolong Liu, Yao Hu, Song Bai, Fei Ding, Xiang Bai, and Philip HS Torr. Multi-shot temporal event localization: a benchmark. In *CVPR*, 2021.

Xiaolong Liu, Song Bai, and Xiang Bai. An empirical study of end-to-end temporal action detection. In *CVPR*, 2022.

Yuan Liu, Lin Ma, Yifeng Zhang, Wei Liu, and Shih-Fu Chang. Multi-granularity generator for temporal action proposal. In *CVPR*, 2019.

Fuchen Long, Ting Yao, Zhaofan Qiu, Xinmei Tian, Jiebo Luo, and Tao Mei. Gaussian temporal awareness networks for action localization. In *CVPR*, 2019.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

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.

Zheng Shou, Dongang Wang, and Shih-Fu Chang. Temporal action localization in untrimmed videos via multi-stage CNNs. In *CVPR*, 2016.

Deepak Sridhar, Niamul Quader, Srikanth Muralidharan, Yaoxin Li, Peng Dai, and Juwei Lu. Class semantics-based attention for action detection. In *CVPR*, 2021.

Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *CVPR*, 2018.

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

Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *ECCV*, 2016.

Limin Wang, Yuanjun Xiong, Dahua Lin, and Luc Van Gool. Untrimmednets for weakly supervised action recognition and detection. In *CVPR*, 2017.

Huijuan Xu, Abir Das, and Kate Saenko. R-C3D: Region convolutional 3D network for temporal activity detection. In *ICCV*, 2017.

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

Mengmeng Xu, Juan-Manuel Pérez-Rúa, Victor Escorcia, Brais Martinez, Xiatian Zhu, Bernard Ghanem, and Tao Xiang. Boundary-sensitive pre-training for temporal localization in videos. In *ICCV*, 2021a.

Mengmeng Xu, Juan Manuel Perez Rua, Xiatian Zhu, Bernard Ghanem, and Brais Martinez. Low-fidelity video encoder optimization for temporal action localization. In *NeurIPS*, 2021b.

Min Yang, Guo Chen, Yin-Dong Zheng, Tong Lu, and Limin Wang. Basictad: an astounding rgb-only baseline for temporal action detection. *arXiv preprint arXiv:2205.02717*, 2022.

Runhao Zeng, Wenbing Huang, Mingkui Tan, Yu Rong, Peilin Zhao, Junzhou Huang, and Chuang Gan. Graph convolutional networks for temporal action localization. In *ICCV*, 2019.

Chenlin Zhang, Jianxin Wu, and Yin Li. Actionformer: Localizing moments of actions with transformers. In *ECCV*, 2022.

Chen Zhao, Ali K Thabet, and Bernard Ghanem. Video self-stitching graph network for temporal action localization. In *ICCV*, 2021.

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

Peisen Zhao, Lingxi Xie, Chen Ju, Ya Zhang, Yanfeng Wang, and Qi Tian. Bottom-up temporal action localization with mutual regularization. In *ECCV*, 2020.

Y Zhao, B Zhang, Z Wu, S Yang, L Zhou, S Yan, L Wang, Y Xiong, D Lin, Y Qiao, et al. Cuhk & ethz & siat submission to activitynet challenge 2017. *CVPR ActivityNet Workshop*, 2017.

# A ARCHITECTURE OF ETAD

As the extension of Section 3.1, here we describe the detailed designs of two modules in ETAD: feature enhancement module and proposal evaluation module. Then, we further introduce the loss function of ETAD.

## A.1 FEATURE ENHANCEMENT MODULE

Long-range relation modeling is vital for temporal action detection (Buch et al., 2017; Khojasteh et al., 2022; Qing et al., 2021), considering that local aggregation alone such as convolution is not sufficient when an action is longer than the receptive field, or correlations between different action instances are critical for prediction (Xu et al., 2020). Although transformers or graph convolutions can also capture temporal context, they usually need more data to converge and strong regularization to optimize. In our work, to enrich the feature representation with global and local information, we adopt LSTM (Hochreiter & Schmidhuber, 1997) to achieve temporal context aggregation.

Figure 5 illustrates the detailed structure of our feature enhancement module. First, a convolution layer with kernel size 3 and output channel 256 is adopted on the video feature sequence. Then a branch consisting of an LSTM layer with hidden channel 256 followed by a convolution layer with kernel size 3 and output channel 128 is used for capturing long-range temporal forward information. Similarly, to obtain the backward information, we flip the input of this branch and feed it into the LSTM layer, then flip the output of this branch again. Next, we concatenate the outputs of these two branches addition with a residual connection which passes a convolution layer with kernel size 256, and get the feature sequence with the channel number of 512. The residual connection can mitigate the effect of forgetting issue brought by the LSTM. Finally, we use a convolution layer with kernel size 1 to downsample the dimension of feature sequence from 512 to 256. Group normalization layer with group number 16 and ReLU are used after each convolution layer. Compared with only local aggregation, our LSTM-based temporal aggregation effectively enhances snippet feature representations in reference to their order, which provides smoother but more distinctive features.

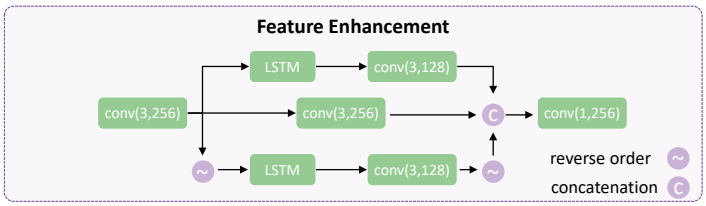



Figure 5: Architecture of feature enhancement module.

**Discussions**: Here we briefly discuss the motivation to use forward and backward aggregation. The disadvantage of capturing long-range information only in one direction, such as forward, is that the snippet feature of the early temporal point can never aggregate the context of the later temporal point, since LSTM processes the snippet features step-by-step in the temporal order. Besides, early temporal context would gradually be forgotten if the neurons move to later temporal point. To overcome this disadvantage, in our work, we use two branches with different aggregation directions which are complementary to each other.

## A.2 PROPOSAL EVALUATION MODULE

The aim of the proposal evaluation module is to refine candidate proposals, which proves to be essential to achieving higher performance as mentioned in Section 2. First, given proposals sampled by detector proposal sampling (DPS), the corresponding proposal features can be extracted on top of the feature sequence. To be more specific, for a proposal $(t_s, t_e)$, we use the interpolation and rescaling algorithm in G-TAD (Xu et al., 2020) as RoI alignment to obtain the proposal start feature, end feature, and extended feature, with the corresponding temporal region of $(t_s - 2/d, t_s)$, $(t_e, t_e + 2/d)$, and $(t_s - 2/d, t_e + 2/d)$, respectively, where $d = t_e - t_s$.

With these proposal features, we further refine the proposals from five aspects: start offset regression, end offset regression, center/width regression, start/end classification, and IoU regression, which is detailed shown in Table 6. For the proposal start feature, several sequential FC layers are adopted to regress the start boundary offset with the nearest ground truth. This is the same for the proposal end feature. For the proposal extended feature, since it covers the context outside the proposal, it is more suitable to regress the IoU between the proposal with the nearest ground truth, which is called proposal IoU regression in Table 6. This IoU score is also used as proposal confidence for ranking. We also regress the offset of the center and width between the proposal and nearest ground truth by several FC layers. What's more, we find that an additional branch to classify the proposal startness and endness by proposal extended feature is helpful for IoU regression. After the proposal evaluation module, the candidate proposals will be refined by the average of start/end offset and center/width offset, which is similar to (Qing et al., 2021).

To further improve the boundary precision of predict actions, we propose the cascade strategy that stacks multiple proposal evaluation module to progressively refine proposal boundaries. To be specific, we sequentially run three stages, 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. Three cascade proposal evaluation modules can not only refine the action proposal boundaries for proposal localization, but they also provide a more reliable IoU score for better proposal ranking. It is worth mentioning that our cascade proposal refinement does not rely on an additional proposal generation network, which is different from Qing et al. (2021).

Table 6: The detailed architecture of proposal evaluation module. $N$ is the number of candidate action proposals.

**Proposal Start Offset Regression**

| layer | dim | act | output size |
|---|---|---|---|
| proposal start 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 End Offset Regression**

| layer | dim | act | output size |
|---|---|---|---|
| proposal 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 Start/End Classification**

| 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 | $sigmoid$ | $2 \times N$ |

**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 | $sigmoid$ | $2 \times N$ |

### A.3 Loss function.

For our end-to-end training, we directly compute our loss function based on the original video frames $V$ (in contrast to snippet features) as well as ground-truth labels $\Psi$, formulated as $\mathcal{L}(V, \Psi)$. The loss function is composed of losses from multiple tasks, including the boundary evaluation loss as well as the cascade proposal refinement loss. $\mathcal{L}$ is computed as follows:

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

where $i$ is the stage index of the cascade proposal evaluation module, and $\lambda_a, \lambda_b, \lambda_c, \lambda_d$ are the weights for the corresponding losses.

$\mathcal{L}_{ce:bd_s}$ in the boundary evaluation module uses batch-level positive-negative-balanced cross entropy to supervise the startness or endness of each snippet. In Eq. 2, $p_{bd,i}$ is the prediction of the startness and endness of each time point $i$, and $b_i$ is the positive mask indicating whether the current time point is boundary start or end. $N_+$ is the number of positive sample in current mini-batch and $N_-$ is the number of negative sample, and $N = N_+ + N_-$.

$$\mathcal{L}_{ce:bd_s} = \frac{N}{2N_+} \sum_i b_i \log p_{bd,i} + \frac{N}{2N_-} \sum_i (1 - b_i) \log (1 - p_{bd,i}). \tag{2}$$

Similar to above, $\mathcal{L}_{ce:bd_p}$ in proposal evaluation module computes the cross entropy for the startness and endness of each proposal's boundaries. We use the same loss function in Eq. 2, but change the $p_{bd,i}$ from the prediction of startness of the time $i$, to the prediction of startness of proposal $i$. This branch is helpful for stabilizing the learning of IoU confidence.

$\mathcal{L}_{iou}$ contains a classification loss and a regression loss for the IoU between prediction and ground-truth. The classification loss is the simple cross entropy loss, whose positive samples are defined with the IoU threshold larger than 0.9. Since the proposal set always have more negative samples, we randomly select the same number of negative samples with the positive samples for balance. As for regression loss, we follow the BMN and use L2 loss to supervise. In Eq. 3, $p_{iou,i}$ is the prediction of IoU confidence for proposal $i$, and $gt_{iou,i}$ is the maximum IoU between the ground truth actions with $i$-th proposal.

$$\mathcal{L}_{iou} = \sum_i \mathcal{L}_{CE}(p_{iou,i}, gt_{iou,i}) + \sum_i \mathcal{L}_2(p_{iou,i}, gt_{iou,i}) \tag{3}$$

For $\mathcal{L}_{rg:secw}$, we use 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, such as 0.7, 0.8, 0.9. In Eq. 4, $p_{c,i}$ is the prediction of the center offset, and $gt_{c,i}$ is the corresponding center offset of the ground truth action closest to i-th proposal. Other regressions follows the same definition.

$$\begin{aligned} \mathcal{L}_{rg:secw} = \sum_{i \in N_+} & \mathcal{L}_{smoothL1}(p_{s,i}, gt_{s,i}) + \mathcal{L}_{smoothL1}(p_{e,i}, gt_{e,i}) \\ & + \mathcal{L}_{smoothL1}(p_{c,i}, gt_{c,i}) + \mathcal{L}_{smoothL1}(p_{w,i}, gt_{w,i}) \end{aligned} \tag{4}$$

# B Implementation Details

## B.1 Data Preparation

**Pre-extracted feature setting.** Here, only pre-extracted snippet features are used, *i.e.* no end-to-end-training is performed. For ActivityNet-1.3, we adopted three types of pre-extracted features: TSN (Wang et al., 2016), TSM (Lin et al., 2019a), and R(2+1)D (Alwassel et al., 2021). Optical flow feature is also concatenated in TSN features. For THUMOS-14, we adopted the TSN features provided by Xu et al. (2020) and I3D features provided by Zhao et al. (2020), which both also use optical flow to extract video features. 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 .

**End-to-end training.** We choose TSM (Lin et al., 2019a) and R(2+1)D (Tran et al., 2018) as our feature encoder for the end-to-end training. We fix the weights of the first two stages of the backbone network and freeze all batch normalization layers of the feature encoder. For TSM, the image resolution is set to 224 and clip length is set to 8, which is the same as in Xu et al. (2021b). For R(2+1)D, the image resolution is set to 112, and clip length is set to 16, which is the same as in Alwassel et al. (2021). We only adopt random cropping as data augmentation. Note that the TSM model is only pretrained on Kinetics-400 (Kay et al., 2017) and not finetuned on the target datasets, *i.e.* ActivityNet-1.3, HACS, or THUMOS-14. The R(2+1)D model is pretrained on the ActivityNet dataset by (Alwassel et al., 2021).

### B.2 TRAINING AND INFERENCE

**Training.** On ActivityNet-1.3 and THUMOS14, we both use a batch size of 16 and the AdamW optimizer (Loshchilov & Hutter, 2019) with weight decay of $10^{-4}$. The learning rate for the action detector is set to $10^{-3}$. In end-to-end training, the learning rate for the feature encoder is set to $10^{-6}$ in TSM and $10^{-7}$ in R(2+1)D. The total training epoch is set to 6 and the learning rate decays by 0.1 after 5 epochs. $\lambda_a, \lambda_b, \lambda_c$ in the loss function are set to 1, while $\lambda_d$ is set to 10.

**Inference.** To post-process network outputs, we use the boundary selecting method in Lin et al. (2019b) to select proposals with high startness and endness, and use the averaged proposal boundary generated from three proposal evaluation modules. We adopt NMS for all predictions by their confidence scores computed as $p = p_s \cdot p_e \cdot p_{iou}$, where $p_s$ and $p_e$ are from $\mathcal{L}_{ce:bd_s}$ standing for the start and end probabilities of a proposal respectively, and $p_{iou}$ is the score of the proposal from $\mathcal{L}_{iou}$. Following Lin et al. (2019b); Qing et al. (2021), we apply the video-level classification scores from Zhao et al. (2017) on ActivityNet-1.3 and Wang et al. (2017) on THUMOS-14. Note that the proposal sampling ratio is set as 100% in testing. Please refer to Appendix C.3 for the ablation of DPS ratio during inference.

## C ADDITIONAL ABLATIONS ON ACTION DETECTOR

In this section, we further conduct more ablation studies on ETAD. The experiments in this section are all under per-extracted feature setting.

### C.1 LSTM-BOOSTED TEMPORAL AGGREGATION

We first study the effectiveness of our feature enhancement module with LSTM-boosted temporal aggregation by comparing it to a convolutional network (Conv) and a vanilla transformer. In Table 7 , Conv shows the lowest run-time and memory usage[1], but also the lowest performance due to its inability to capture long-range context. Vanilla transformer takes the most resources without bringing mAP gains. Comparatively, our LSTM-based module requires a moderate amount of memory, while achieving high performance without the requirements of more data to converge and strong regularization to optimize as with transformers.

Table 7: Study of different feature enhancement modules of proposed action detector on ActivityNet-1.3 with TSN features. Only one proposal evaluation stage is used in this ablation.

| Feature Module | Training Time | Memory (GB) | Avg. mAP |
|---|---|---|---|
| Conv | 2min 35s | 0.47 | 34.80 |
| Transformer | 3min 05s | 0.68 | 34.82 |
| LSTM | 3min 23s | 0.53 | **35.25** |

### C.2 CASCADE PROPOSAL REFINEMENT

We also study the impact of the number of stages used for the cascade proposal refinement, as shown in Figure 8. With a single proposal evaluation stage, the average mAP is 35.25%, and performance consistently improves as we use more proposal evaluation modules. Considering efficiency as well, we choose to use three stages in total and achieve an average mAP of 36.06%.

### C.3 DPS DURING INFERENCE

As described in the paper, proposal-level sampling is adopted in ETAD to reduce the training computation cost, *i.e.* DPS. As the default, we only perform DPS during training. In inference, we

[1]Following mmaction2 (Contributors, 2020), we use `torch.cuda.max_memory_allocated()` to compute the GPU memory.

Table 8: Study of different cascade stages of proposed action detector on ActivityNet-1.3 with TSN features.

| Cascade Stages | Training Time | Memory (GB) | Avg. mAP |
|---|---|---|---|
| 1 | 3min 23s | 0.53 | 35.25 |
| 2 | 4min 33s | 0.88 | 35.81 |
| 3 | 5min 58s | 1.23 | **36.06** |

use all the predicted proposals for higher detection performance. However, as the tool for generating proposals, DPS can also be applied during inference. Based on such motivation, we adopt grid sampling strategy with DPS during inference, and the results (Table 9) show that DPS is also effective for reducing inference complexity while preserving accuracy. Only the sampling ratio is smaller than 15%, the detection starts to decrease visibly. We did not conduct DPS in inference as the default, considering that the inference time is rather small compared to training time.

Table 9: Ablations of DPS during inference on ActivityNet-1.3 with TSN features. Post-processing time is included in the inference time.

| DPS Ratio | 100% | 20% | 15% | 10% | 6% |
|---|---|---|---|---|---|
| mAP | 36.06 | 35.97 | 35.94 | 35.86 | 35.66 |
| GFLOPs (per video) | 180.08 | 36.60 | 27.63 | 18.66 | 11.49 |
| Inference Time (per video) | 21.3ms | 10.7ms | 10.3ms | 10.1ms | 9.9ms |

## D RESULTS ON HACS DATASET

We also report our results on HACS (Zhao et al., 2019) dataset based on the pre-extracted features. 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 (Feichtenhofer et al., 2019) features provided by Qing et al. (2021) and rescale the feature sequences to 224 snippets. The only training difference from ActivityNet is that we use the learning rate of $4 \times 10^{-4}$ on HACS.

As shown in Table 10, ETAD can outperform the baseline method BMN by a large margin. Compared with state-of-the-art method TCANet (Qing et al., 2021), ETAD can also achieve comparable performance. However, 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. ETAD also exceed TCANet on the high IoU threshold scenario by 2.5%, which is similar as in ActivityNet-1.3 and THUMOS14. 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. Besides, we also test different detector 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. The above results prove the effectiveness of our ETAD on this larger dataset.

Table 10: Comparison of ETAD with other methods on HACS with SlowFast features.

| Methods | 0.5 | 0.75 | 0.95 | Avg. mAP | Memory (GB) | Training Time |
|---|---|---|---|---|---|---|
| BMN | 52.49 | 36.38 | 10.37 | 35.76 | 12.10 | 58 min |
| BMN+TCANet | 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 |

## E END-TO-END TRAINING ON THUMOS DATASET

To further verify the effectiveness of proposed encoder gradient sampling, we also conduct end-to-end training on THUMOS dataset, as shown in Table 11. In this experiment, we choose SlowOnly-ResNet50 or TSM-ResNet50 as the feature encoder, and only use RGB modality without optical flow. The clip length of snippets is set as 8, and frame resolution is $180\times180$. Only random cropping is adopted, and we train our model with 6 epochs. Since we are using shorter clip, lower resolution and only single modality, the performance is expected to be lower than state-of-the-art method.

From the results, we can find that if the encoder gradient sampling ratio ($\gamma$) is 0, *i.e.* pre-extraced feature setting, the performance on THUMOS 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 features, and more than 5% gains of average mAP using TSM features. This verifies the importance of end-to-end training again. Besides, with different sampling ratios, we interestingly find the performances remain at the same level, suggesting that a small portion of snippets is enough for end-to-end training in TAD. Moreover, with our EGS, we successfully reduce the computation cost from 34.6GB to 4.1GB per video (cutting down 89%), but maintain the same detection performance. This further proves the existence of snippet-level redundancy in end-to-end learning, and also shows the effectiveness of EGS method.

Table 11: Performance of end-to-end training on THUMOS test dataset. $\gamma$ is the encoder gradient sampling ratio. Only RGB modality is adopted.

| Feature Encoder | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | Avg. mAP | Memory (GB) |
|---|---|---|---|---|---|---|---|
| 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 | 3.7 |
| SlowOnly ($\gamma=30\%$) | 60.66 | 52.87 | 42.95 | 33.31 | 23.38 | 42.63 | 6.9 |
| SlowOnly ($\gamma=100\%$) | 60.18 | 52.93 | 44.40 | 33.88 | 23.76 | 43.03 | 19.6 |
| 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 | 4.1 |
| TSM ($\gamma=30\%$) | 56.50 | 49.16 | 39.17 | 29.47 | 19.07 | 38.67 | 8.4 |
| TSM ($\gamma=100\%$) | 57.44 | 48.99 | 39.55 | 29.37 | 18.60 | 38.79 | 34.6 |

## F 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. Note that encoder gradient sampling ratio is set to 100% in the following experiment, which means the completely end-to-end training is applied for following ablation studies.

**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 traditional feature-based methods. As shown in Table 12, we implement random cropping and temporal jittering at 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, demonstrating the superiority of end-to-end training. However, it seems that temporal jittering slightly harms the performance. Therefore, we only use random cropping in our experiment as default.

Table 12: Study of different data augmentation in end-to-end training on ActivityNet-1.3.

| Backbone | Frame Resolution | Data Augmentation | Frozen Stage | Average mAP |
|---|---|---|---|---|
| TSM | 112x112 | × | 2 | 35.12 |
| TSM | 112x112 | jitter | 2 | 35.17 |
| TSM | 112x112 | crop | 2 | **35.53** |
| TSM | 112x112 | crop+jitter | 2 | 35.38 |

**Partially freeing backbone can have a good trade-off between computation and performance.** It is a common trick to save the GPU memory by freezing some shallow layers of encoder backbone. 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. In Table 13, the frozen stage of 4 means we freeze all the backbone layers, indicating the encoder will not be updated and it will degenerate into none-end-to-end training. And frozen stage 0 means we do not freeze any layers and the encoder will be optimized in completely end-to-end training, however the memory consumption is also the largest as expected. Table 13 clearly shows that: **1)** End-to-end training is important for temporal action detection, since the frozen stage of 4 has the lowest detection performance compared with others. **2)** As we freeze fewer encoder layers, detection performance will be improved, but the gain becomes smaller. **3)** To have a good trade-off between memory and performance, frozen stage 2 is recommended in our experiments.

Table 13: Study of different frozen stage, frame resolution and pretraining of backbone in end-to-end training on ActivityNet-1.3. We report the GPU memory usage per video. † means out of memory on 8 A100 GPUs. ‡ means the encoder is finetuned on ActivityNet by the classification task.

| Backbone | Frame Resolution | Data Augmentation | Frozen Stage | Average mAP | Memory (GB) |
|---|---|---|---|---|---|
| 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 |

**Higher frame resolution can boost the performance by a large margin.** In the traditional feature extraction process in TAD, to pursue a high-quality action detection performance, we tends to use higher image resolution, and denser temporal sampling rates. Similarly, we also study the impact of the frame resolution on the detection performance in end-to-end training, as shown in Table 13. If we freeze all the backbone layers, a higher resolution can bring 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. However, since both freezing fewer layers and using higher frame resolution will cost much more GPU memory, we recommend giving priority to higher frame resolution, since it can bring more performance gains.

**Classification pretraining is not necessary if adopt end-to-end training.** Some other TAD methods proposed to finetune the video encoder on the target dataset by the classification task, which means the encoder will be trained to classify the video clips in the annotated action instances. In our end-to-end training experiment, if we just use the model pretrained on Kinetics-400, we can have a 36.79 mAP. If we finetune the model on the ActivityNet dataset by a classification task, the detection performance would be slightly improved to 36.92 (see Table 13). Such small gain (+0.13) is reasonable, since end-to-end training can improve the feature representation ability from target dataset as much as possible. However, since the classification pretraining stage also takes long time to converge, 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 to adopt end-to-end training. If necessary, we can further freeze certain layers in the backbone. We believe such a study for end-to-end training will enlighten the TAD community in the sense of efficiency and efficacy trade-off.

## G IMPLEMENTATION OF ENCODER GRADIENT SAMPLING

In this section, we briefly describe the implementation of Encoder Gradient Sampling (EGS). The detailed procedure can be described in algorithm 1. To implement EGS, we first split (*e.g.* through random sampling) the video snippets into two sets: with gradient snippets $X_a$ and without gradient snippets $X_b$. Then, we feed $X_b$ to the feature encoder without saving gradients to produce the corresponding features $F_b$. For $X_a$, we feed them into the encoder to produce features $F_a$, while preserving the encoder's gradients when backpropagating. We then concatenate $F_a$ and $F_b$ together and arrange them by their original order as feature sequence $F$, which is sent to the efficient action detector. During backpropagation, the encoder parameters are only updated by the gradient given by $X_a$. Also, we did not observe that such sampling harms the stability of the network training.

**Algorithm 1** Encoder Gradient Sampling

**Input:** sampling ratio $\gamma$; video snippets $X \in \mathbb{R}^{M\times 3\times T\times H\times W}$, where $M$ is the snippet number, $T$ is the frame number of each snippet;

1: Select $M_a$ snippets from $X$ on dimension $M$ based on the sampling ratio $\gamma$, denoted as $X_a$. The unselected snippets is denoted as $X_b$. Keep the selection index $idx$ for restoring the snippet order later. ($X_a \in \mathbb{R}^{M_a\times 3\times T\times H\times W}$, $X_b \in \mathbb{R}^{M_b\times 3\times T\times H\times W}$, $M = M_a + M_b$, $M_a = \gamma \times M$)
2: Turn off the gradient computing of the video encoder. Feed $X_b$ to the video encoder and obtain the feature vector $F_b$. Clear the CUDA cache. ($F_b \in \mathbb{R}^{M_b\times C}$, $C$ is the channel number.)
3: Turn on the gradient computing of video encoder. Feed the $X_a$ to the video encoder and obtain the feature vector $F_a$. ($F_a \in \mathbb{R}^{M_a\times C}$)
4: Concatenate $F_a$ and $F_b$ together, and use selection index $idx$ to restore the orignial snippet order, denoted as $F \in \mathbb{R}^{M\times C}$.
5: Feed $F$ to the following action detector.
6: Compute the gradients of the video encoder and action detector by loss function.
7: Update the parameters of video encoder and action detector by optimizer. The parameters of video encoder is only updated on the selected data, which is $X_a$.

**Output:** Updated video encoder and action detector.

In addition, in our implementation with PyTorch1.12 and CUDA11.1, we find that the tensor size is limited to $2^{31}$ by the platform. Therefore, given a small sampling ratio such as 10%, the size of unsampled data $X_b$ may be larger than the above limitation. So we divide the $X_b$ into smaller chunks on snippet dimension and feed them to the video encoder step-by-step. After feature extraction, we concatenate the corresponding features together. As for $X_a$, thanks to our EGS, the tensor size is usually smaller than the limitation, which does not need extra processing.

## H DETAILS OF FEATURE-GUIDED SAMPLING STRATEGY

Beyond the naive heuristic samplings (*random*, *grid*, *block*) that the sampling process is irrelevant to the data itself, we also explore feature-guided sampling, *e.g. farthest point sampling*, *determinantal point process*. The motivation of feature-guided sampling is that each data has different features than others, which means their inherent importance are also different with others. Therefore, one can try to leverage the information inside the data and ensure the most informative or important samples are selected. This is different from heuristic sampling since the sampling process of feature-guided sampling performs on the embedding space of samples (*e.g.* snippet features, proposal features). The pseudocode of FPS/DPP can be found in algorithm 2.

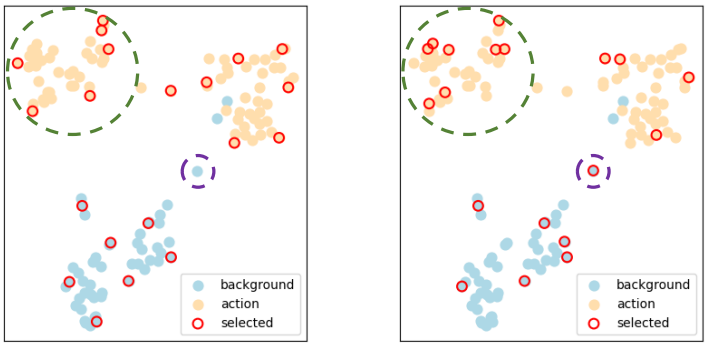
The farthest point sampling (FPS) is a common feature-guided sampling approach, which has been adopted in many fields such as point cloud understanding. Given the data points $X \in \mathbb{R}^{N\times C}$, where $N$ is the number of total samples, and $C$ is the dimension number of each sample features, FPS selects the new point from the unselected points, and ensures new new point has the farthest distance to the current 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 sample 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: (string) fps or dpp.
import torch_cluster
from dppy.finite_dpps import FiniteDPP

# 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 * data.shape[0])
   # likelihood kernel, use eye matrix to increase the rank
   kernel = data.dot(data.T) + 1e-2 * np.eye(data.shape[0])
   DPP = FiniteDPP("likelihood", **{"L": kernel})
   index = DPP.sample_exact_k_dpp(size=sample_num))

# return the index of selected samples (list, 0~N)
```

We also implement another feature-guided sampling as the determinantal point process (DPP). DPP measures the sample probability as a determinant of some functions or 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 (Kulesza & Taskar, 2011) 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 likelihood matrix is larger than the sample size. In general, DPP sampling improves the diversity of sampled data in the embedding space.

In our experiments, we find that DPP works always better than FPS. To further analyze these two strategies, we provide the t-SNE visualization (Van der Maaten & Hinton, 2008) of FPS and DPP at the snippet level, as shown in Figure 6. In this figure, the yellow points are the action foreground snippets, and the blue points are the background snippets. We leverage the snippet feature before classification head in the feature encoder to adopt t-SNE. 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 dash green circle, FPS tends to select extreme points, while DPP can select samples with larger variety. **(2)** For the point in the purple dash circle, FPS misses such hard-negative sample because its distance with other points is not that big in the embedding space. However, such points may be informative samples, and DPP successfully select this representative sample. Those two findings can explain the success of DPP sampling.




(a) FPS (b) DPP

Figure 6: t-SNE visualization of FPS sampling and DPP sampling.

# I QUALITATIVE VISUALIZATION

In order to provide a more vivid understanding of our method, we visualize the qualitative predictions of our method and BMN (Lin et al., 2019b) on ActivityNet for comparison. In Figure 7, we plot the ground truth actions of each video (drawn in black and above the black line), and also 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.

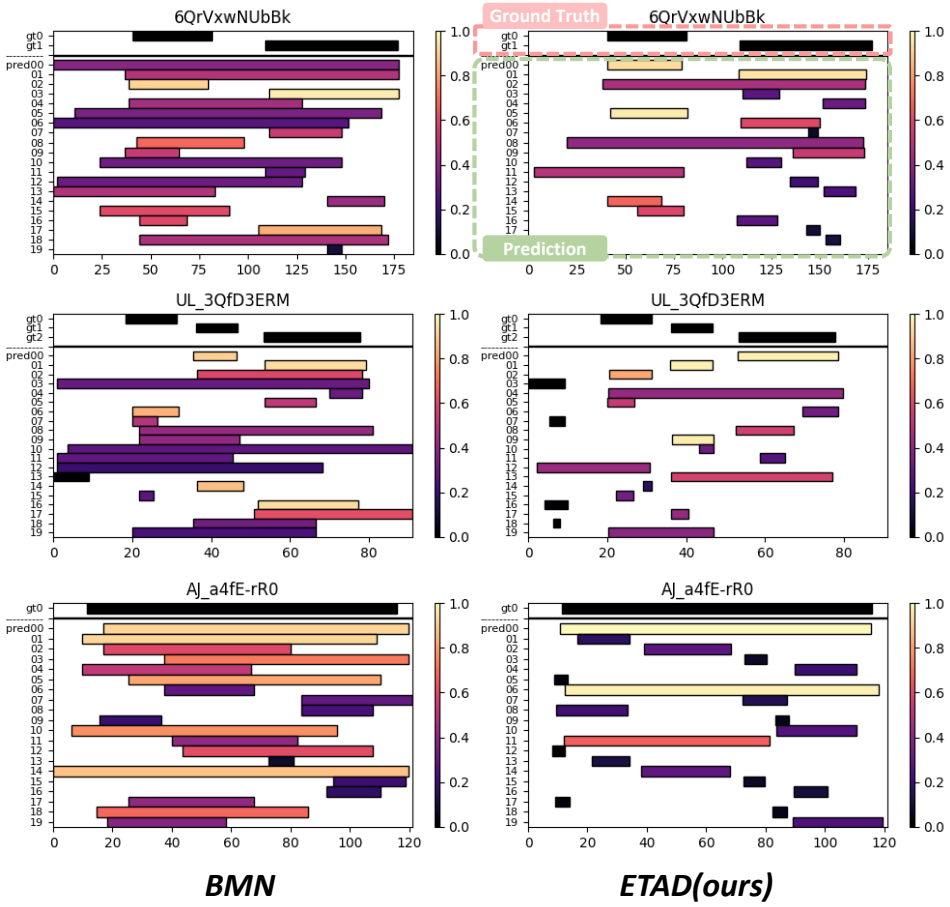



Figure 7: Qualitative results of ETAD and BMN on ActivityNet-1.3. The color of the proposal represents the maximum IoU of this proposal to ground truth actions. We plot the ground truth actions of each video (drawn in black and above the black line), and top-20 predicted proposals by algorithms (drawn in colors and under the black line).

As demonstrated in the figure, ETAD can generate **(1) more precise proposal boundary.** For instance, in the first and third row in Figure 7, 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 Figure 7, 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.