

Appendix

A Detailed Descriptions

A.1 Model Configuration

Sampler. The sampler S_{SO} takes a sequence of T frames as input, composed of a lightweight feature extractor f_s , an importance predictor h_s , and an action classifier h_c .

Given T candidate frames $\mathbf{v} \in \mathbb{R}^{T \times 3 \times H \times W}$, our sampler first spatially downsamples them to $\mathbf{v}' \in \mathbb{R}^{T \times 3 \times H' \times W'}$, where $W' < W$ and $H' < H$. Then, a 2D image representation network $f_s : \mathbb{R}^{3 \times H' \times W'} \rightarrow \mathbb{R}^D$ extracts frame-level features, where D denotes the dimensionality of the feature. Inferring all T frames using f_s , a feature map

$$\mathbf{z} = \{f_s(\mathbf{v}'_1), \dots, f_s(\mathbf{v}'_T)\} \in \mathbb{R}^{T \times D} \quad (5)$$

is constructed, where \mathbf{v}'_i denotes the i -th frame of \mathbf{v}' .

Then, our sampler conducts two downstream tasks using the extracted features \mathbf{z} . First, it estimates the frame importance score \mathbf{p}_s using a regressor $h_s : \mathbb{R}^{T \times D} \rightarrow \mathbb{R}^T$:

$$\hat{\mathbf{p}}_s = \{h_s(\mathbf{z}_1), \dots, h_s(\mathbf{z}_T)\}. \quad (6)$$

Second, it performs the downstream classification task. Using a frame-level classifier $h_c : \mathbb{R}^{T \times D} \rightarrow \mathbb{R}^C$, it predicts the relevance of each frame $t = 1, \dots, T$ for the C classes, and these predictions are aggregated over the T frames to a video-level prediction by taking the average:

$$\hat{\mathbf{y}}_s = \frac{1}{T} (h_c(\mathbf{z}_1) + \dots + h_c(\mathbf{z}_T)). \quad (7)$$

Note that h_c is used only during training to make the backbone f_s learn the label information. We simply implement h_s and h_c with linear projections followed by a softmax.

Classifier. The classifier f_c can be any visual recognition model, such as a 2D or 3D CNN, or a Transformer, pretrained and frozen throughout the training. During training, f_c is used to compute the importance score, which serves as a pseudo-label to train the sampler S_{SO} by distilling the knowledge from the classifier f_c .

During inference, f_c is used to perform the downstream task on a clip $\mathbf{v}^s \in \mathbb{R}^{N \times 3 \times H \times W}$ of sampled N frames:

$$\hat{\mathbf{y}} = f_c(\mathbf{v}^s). \quad (8)$$

Our goal is to train the sampler S_{SO} so that the frozen classifier f_c predicts $\hat{\mathbf{y}}$ close to the ground truth label \mathbf{y} , the one-hot encoding of the true label \mathbf{y} .

A.2 Dataset

ActivityNet-v1.3 includes 10,024 training and 4,926 validation videos. The average video length is 117 seconds with an average of 3,335 frames, covering 200 categories. Mini-Kinetics, a subset of Kinetics400, comprises 121,215 training and 9,867 validation videos. The videos have an average duration of 10 seconds with an average of 261 frames, covering 200 categories. Mini-Sports1M, a subset of Sports1M [10], consists of 14,586 training and 4,855 validation videos. The average video length is 330 seconds with an average of 4,467 frames, covering 487 action classes. COIN is composed of 11,827 YouTube videos related to 180 different tasks. The videos have an average length of 141 seconds with an average of 4,009 frames.

Dataset	Backbone	Method	N / T		
			8 / 30	16 / 60	32 / 100
Mini-Kinetics	ResNet	OCSampler	73.52%	74.00%	74.17%
		SOSampler	73.79%	74.46%	74.65%
	TimeSformer	OCSampler	79.13%	78.05%	76.33%
		SOSampler	79.93%	80.44%	81.32%
COIN	ResNet	OCSampler	78.69%	79.79%	80.06%
		SOSampler	79.13%	80.21%	81.02%
	TimeSformer	OCSampler	80.88%	80.90%	81.20%
		SOSampler	86.52%	87.37%	88.08%

Table I: **Experiment on short video datasets for large N and T .** The best performing model is **bold-faced**.

A.3 Implementation Details

We follow the preprocessing steps outlined in [24]. We sample T frames from each video as a training example. All frames are randomly scaled and cropped to 224×224 , followed by random flipping for augmentation. We then reduce the resolution of each frame to 128×128 before feeding them into our sampler S_{SO} . During inference, we uniformly sample T frames from a test video, resize them to 128×128 , and feed them to the sampler S_{SO} . Then, we feed the original 224×224 images of the selected frames to f_c .

For the pretrained classifier weights, we utilize the pretrained weights provided by [11] on the ActivityNet-v1.3 and Mini-Kinetics datasets with the ResNet50 classifier. For other datasets and architectures, we train the classifier from scratch.

To train our SOSampler, we use a learning rate of 10^{-3} and set $\lambda = 0.99$ for all datasets. We optimize our loss function in Eq. (4) using the stochastic gradient descent (SGD) optimizer with a momentum of 0.9 and weight decay set to 10^{-4} . We employ cosine annealing as a learning rate scheduler without warm-up.

We implement our method using PyTorch and train on a single NVIDIA A100 GPU with 40GB of memory.

B Additional Results

B.1 Comparison with Large N and T on Short Videos

In short videos, as T increases, the FPS becomes significantly higher, weakening our assumption of independence between frames. Therefore, our approach does not sufficiently improve the performance as N and T increase. However, it consistently shows an upward trend and still outperforms OCSampler in all settings. This result suggests that our method is still superior to the existing method, even in the FPS range where our assumption of independence between frames is weak.

B.2 Computational Efficiency

While GFLOPs serve as a metric for measuring the efficiency of a model, it does not provide the actual running time. Therefore, we additionally compare the actual inference

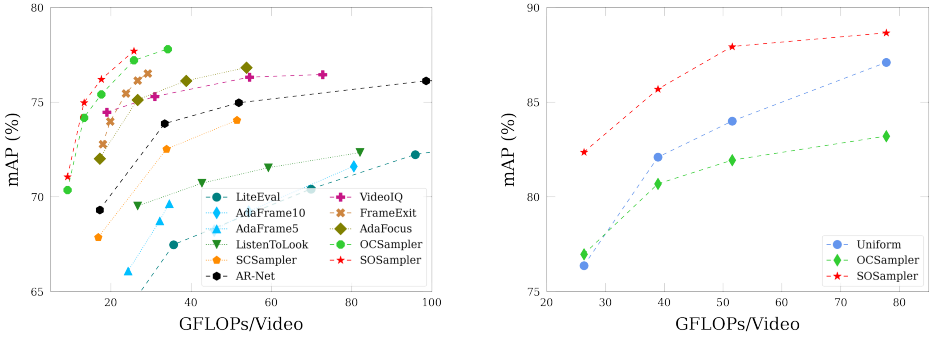


Figure I: **Mean Average Precision (%) vs. efficiency (GFLOPs) on ActivityNet.** With a ResNet classifier (*left*), OCSampler [24] is the second best after ours. With TimeSformer in (*right*), however, it even underperforms than the uniform sampling. On the other hand, our approach outperforms all baselines with both ResNet50 and TimeSformer.

time, namely throughput, by measuring the video processing speed per second. The experiments are conducted using ResNet50 on the ActivityNet-v1.3 dataset, and all experiments are performed on a single NVIDIA Xp GPU. As seen in Tab. II, we demonstrate improved accuracy of our proposed method (SOSampler) over existing methods, achieving a reduction of approximately 16.3% in GFLOPs and a 15% enhancement in throughput.

Methods	mAP	GFLOPs	Throughput (Videos/s)
AdaFrame [44]	71.5%	79.0	6.4
FrameExit [11]	76.1%	26.1	19.1
AR-Net [26]	73.8%	33.4	23.1
AdaFocus [41]	75.0%	26.6	44.9
OCSampler [24]	77.2%	25.8	107.7
SOSampler (ours)	77.3%	21.6	123.9

Table II: **Comparison of computational overhead** in GFLOPs and throughput. (ResNet50 on ActivityNet)

B.3 Performance and Efficiency Curve

In Fig. I(left), we compare our approach to existing methods with varying computational costs, with a varied number of sampled frames $N = 2, 3, 4, 6$ on a ResNet50 [12] classifier. Our method leads all other compared methods, using significantly lower computational cost than most baseline methods, showing marginal improvement over OCSampler [24].

We additionally conduct a performance and efficiency comparison using the TimeSformer [2] backbone. The experiment, like the one performed on ResNet50, measures the changes in computational cost for $N = 2, 3, 4, 6$ and the comparison is exclusively with OCSampler, previously the highest-performing model. As shown in Fig. I(right), within the TimeSformer architecture, our model significantly improves the performance over OCSampler.

C Sampling Cases

In Sec. 4.1, we introduce π_s and demonstrate that it approximates π_o through Tab. 1 and Tab. 2. By showing that SOSampler, which learns π_o instead of π_s , outperforms existing methods across various datasets and architectures, we demonstrate the effectiveness of π_s .

In this section, for a better understanding of our approach, we visually illustrate multiple examples showing that SOSampler successfully approximates π_s as well as some cases where it does not.

C.1 Illustration of Successful Sampling





									
Riding Camel									
π_o	0	0			0	0	0		0
π_s	0	0			0	0	0		0
S_{SO}	0	0			0	0	0		0
									
Washing Dishes									
π_o	0			0	0	0	0	0	
π_s	0			0	0	0	0	0	
S_{SO}	0			0	0	0	0	0	
									
Braiding Hair									
π_o	0	0	0	0			0	0	
π_s	0	0	0	0			0		0
S_{SO}	0	0	0	0			0		0
									
Peeling Potatoes									
π_o		0	0			0	0	0	0
π_s		0	0			0	0	0	0
S_{SO}		0	0			0	0	0	0

Figure II: Sampling policy comparison with π_s , π_o for success cases of SOSampler.

In Fig. II, we present qualitative examples of successful sampling by SOSampler, comparing the results with those of π_o and π_s . For the “Riding Camel” example, although the 4th and 5th frames feature a camel, they are not selected due to the lack of clear information about riding compared to other scenes. In the “Braiding Hair” example, π_o and π_s choose slightly different frames, with SOSampler following the selection pattern of π_s . In the case

of “Peeling Potatoes”, it is observed that all policies effectively sample only the portions where potatoes appear.

These results demonstrate that π_o and π_s possess similar policies. Additionally, they indicate that SOSampler can effectively learn the policy of π_s .

C.2 Failure Cases

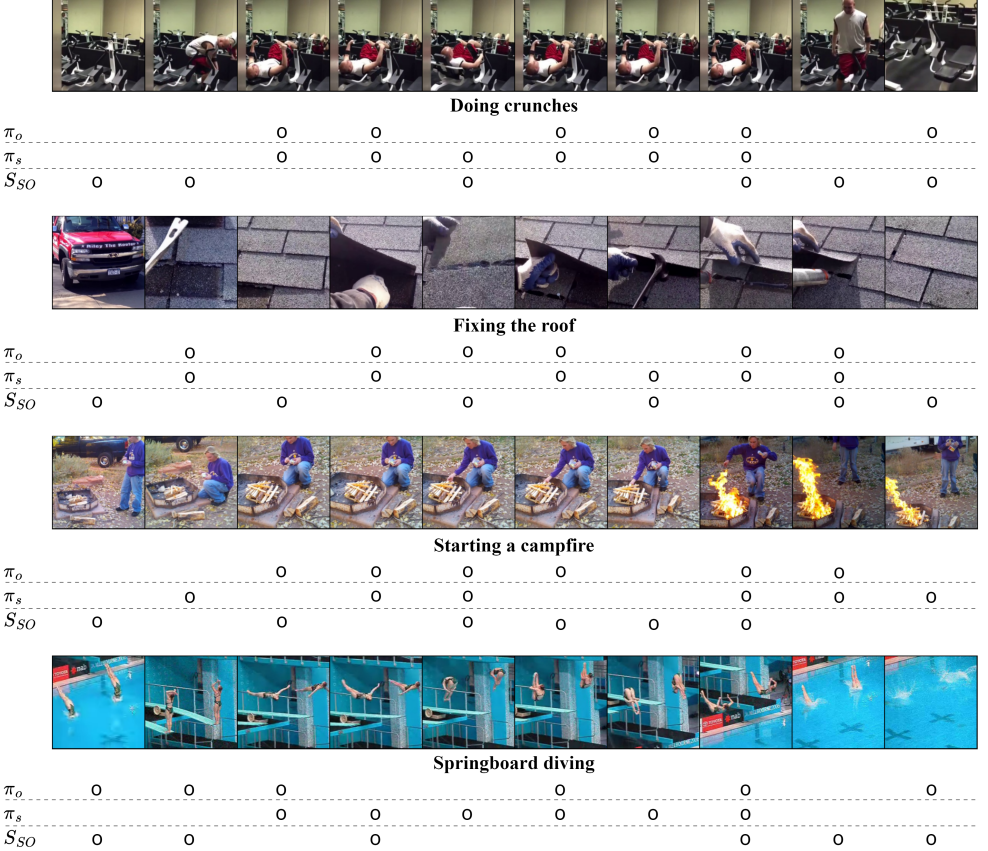


Figure III: Sampling policy comparison with π_s , π_o for failure cases of SOSampler.

As shown in Fig. II, in most cases, S_{SO} demonstrates a sampling policy similar to π_s . In Fig. III, however, we showcase a few scenarios where they significantly differ. In the case of the “Doing Crunches”, π_s effectively samples the segments where a man is performing crunches, while S_{SO} samples scattered frames throughout the video. For the “Fixing the Roof”, π_s appropriately selects scenes of repairing damaged roofs, while S_{SO} chooses unrelated frames as well. In the case of “Starting a Campfire”, S_{SO} seems to summarize the video well, but the sampling policy of π_o indicates that the classifier f_c prefers the scenes of installing firewood and starting the fire. Interestingly, in the “Springboard Diving” example, S_{SO} even appears to better emulate π_o than π_s does.