Cluster-Wise Spatio-Temporal Masking for Efficient Video-Language Pretraining

Supplementary Material

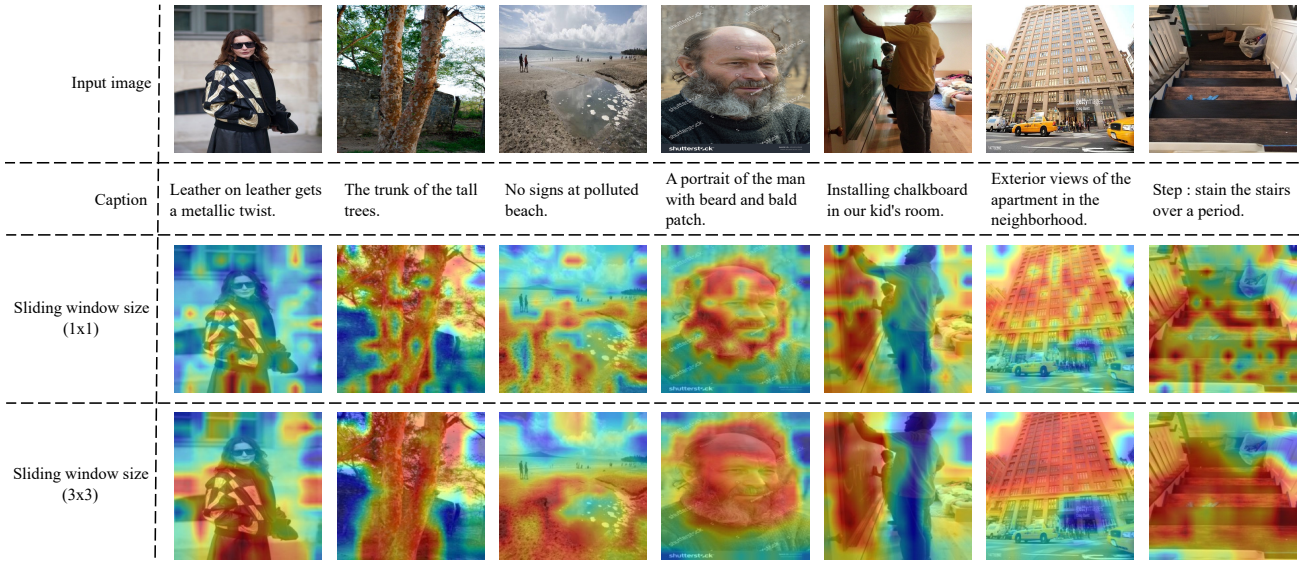

Figure I. Visualization of the relevance heatmaps generated with different sliding window size. From top to bottom, we show the input image, the corresponding caption, the relevance heatmap obtained with a 1x1 sliding window, and the relevance heatmap obtained with a 3x3 sliding window.

A. Additional Analysis

Different Sliding Window Size for Relevance Generation. To investigate the effect of different sliding window sizes on the quality of video-text relevance generation, we generate relevance matrices using sliding windows of size 1x1 and 3x3, respectively. The visualization of the relevance heatmaps are shown in Figure I. As illustrated, directly interacting a single visual token with the text feature fails to produce high-quality relevance scores that accurately capture the correspondence between visual and textual information. This is primarily because the teacher model is trained with global vision-language contrastive learning, which leads it to pay limited attention to local semantic details. In contrast, after aggregating neighboring tokens, the resulting relevance maps exhibit significantly improved quality.

Meanwhile, Table I reports the experimental results obtained with different sliding window sizes across various downstream tasks. As shown, using a 3x3 sliding window leads to substantial performance gains over the 1x1 setting, further demonstrating the effectiveness of our relevance generation strategy.

Table I. Experimental results of different sliding window sizes for relevance generation on MSRVTT [1] dataset across three downstream tasks. The best results are indicated in **bold**.

Sliding Window Size	Retrieval	QA	Captioning
1x1	39.7	40.2	54.7
3x3	**41.3**	**42.7**	**56.2**

Qualitative Results of Cluster-wise Spatio-Temporal Masks. As shown in Figure II, we present the visualization of the masks generated by our cluster-wise spatio-temporal masking strategy. The cluster-wise masking ensures that the retained tokens collectively preserve the complete visual content of the video. Moreover, our masking strategy successfully captures the dynamic motion cues of the *head* and *legs* during the *yoga* action, and, guided by temporal density, retains the tokens that are most semantically correlated over time. As a result, the preserved tokens exhibit strong temporal consistency.

Limitations and Future Work. Although cluster-wise spatio-temporal masking demonstrates promising effectiveness in addressing the severe visual information loss and temporal information leakage encountered when applying

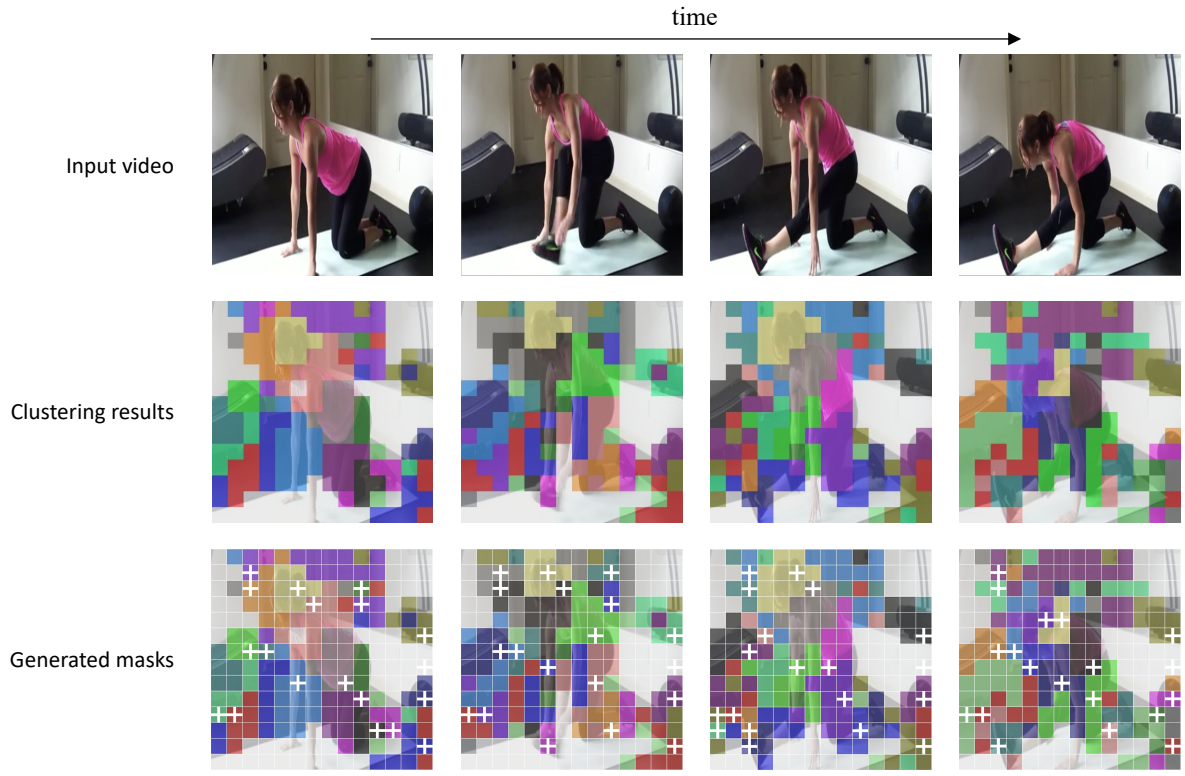

Figure II. Visualization of the cluster-wise spatio-temporal masking. From top to bottom, we show the input video, the clustering results, and the generated masks, where "+" indicates the tokens retained by the masking strategy.

masked visual modeling to large-scale video-language pre-training, it relies heavily on the quality of clustering, and the clustering algorithm we adopt is non-learnable, which constrains its adaptability to diverse video domains and dynamic spatio-temporal patterns. A potential direction for future work is to design learnable spatio-temporal deep clustering algorithms that can be jointly optimized with the pre-training objectives. Such approaches could enable the clustering process to better align with high-level video-language semantics and dynamically adapt to scene variations.

B. More Implementation Details

Pseudocode of Cluster-Wise Spatio-Temporal Masking. Intra-frame clustering divides visual tokens into multiple semantically independent clusters, and the number of tokens within each cluster often varies. This irregularity poses challenges for designing code that can efficiently process multiple video inputs in parallel. Therefore, we provide PyTorch-style pseudocode for parallel cluster-wise spatio-temporal masking, as shown in Algorithms I and II.

Pseudocode of Video-Text Relevance Generation. The usage of sliding windows poses challenges for performing relevance generation in parallel. To address this, we provide PyTorch-style pseudocode for parallel relevance generation, as shown in Algorithm III.

References

[1] Jun Xu, Tao Mei, Ting Yao, and Yong Rui. Msr-vtt: A large video description dataset for bridging video and language. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5288–5296, 2016. 1

Algorithm I PyTorch-style pseudocode for parallel cluster-wise spatio-temporal masking.

```
def ClusterSTM(x, mask_ratio, dc_ratio):
  # x: normalized visual tokens
  # mask_ratio: masking ratio applied for masking
  # dc_ratio: quantile percentage for cutoff distance

  B,T,N,C = x.shape
  num_clusters = N * (1 - mask_ratio)

  #------- Step 1: Density Peak Clusteing (DPC) -------
  clusters = DPC(x, num_clusters)
  clusters = clusters.reshape(B*T,N)

  #------- Step 2: Temporal Density Calculation -------
  temp_rho = TDC(x, dc_ratio)

  #----------- Step 3: Cluster-Wise Masking -----------
  flat_rho = temp_rho.reshape(-1)
  flat_clusters = clusters.reshape(-1)

  # add offsets to cluster IDs
  max_clusters = clusters.max(dim=1).values+1
  clusters_offset = torch.cat([torch.tensor([0]),
       max_clusters.cumsum(0)[:-1]])
  global_offset = clusters_offset.repeat_interleave(N)
  global_clusters = flat_clusters + global_offset

  # cluster-wise softmax
  num_global_clusters = clusters_offset[-1] +
       max_clusters[-1]
  sum_rho = torch.zeros(num_global_clusters).
       scatter_add_(0, global_clusters, flat_rho)
  global_scores = fllat_rho / (sum_rho[global_clusters]
       + 1e-12)

  # cluster-wise argmax
  max_per_cluster = torch.full((num_global_clusters,),
       -1e9)
  max_per_cluster.scatter_reduce_(0, global_clusters,
       global_scores, reduce="amax", include_self=True)

  # masking
  masks = global_scores != max_per_cluster[
       global_clusters]
  masks = masks.reshape(B,T,N)
  return masks
```

Algorithm II PyTorch-style pseudocode for parallel temporal density calculation.

```
def TDC(x, dc_ratio):
  # x: normalized visual tokens
  # dc_ratio: quantile percentage for cutoff distance

  B,T,N,C = x.shape

  spat_x = x.reshape(B*T,N,C)
  temp_x = x.unsqueeze(1).expand(-1,T,-1,-1,-1)
  mask = ~torch.eye(T)
  temp_x = temp_x[:,mask].reshape(B*T,(T-1)*N,C)

  # compute the temporal semantic distances between each
       token and all tokens in neighboring frames
  temp_dist = 1 - torch.matmul(spat_x, temp_x.transpose
       (-1,-2))

  # compute the cutoff distance
  dc = torch.quantile(temp_dist.view(B*T,-1), dc_ratio,
       dim=1, keepdim=True).view(B*T,1,1)

  # compute temporal density based on the temporal
       semantic distances
  temp_rho = torch.sum(torch.exp(-(temp_dist / dc)**2),
       dim=-1) / N
  return temp_rho
```

Algorithm III PyTorch-style pseudocode for parallel relevance generation.

```
def RG(x, y, Pooler):
  # x: visual tokens output by the vision encoder
  # y: text feature output by the text encoder
  # Pooler: pooling module of the teacher model

  B,T,N,C = x.shape
  L = int(N**0.5)

  x = x.permute(0,1,3,2).reshape(B*T,C,L,L) # B*T,C,L,L
  x = torch.nn.functional.pad(x, (1,1,1,1), mode="
       replicate") # B*T,C,L+2,L+2
  x = x.unflod(2,3,1).unfload(3,3,1) # B*T,C,L,L,3,3
  x = x.permute(0,2,3,4,5,1) # B*T,L,L,3,3,C
  x = x.reshape(-1,9,C) # B*T*L*L,9,C
  x = Pooler(x) # B*T*L*L,C
  x = x.reshape(B,T*N,C) # B,T*N,C

  x_norm = x / x.norm(p=2, dim=-1, keepdim=True) # B,T*N
       ,C
  y_norm = y / y.norm(p=2, dim=-1, keepdim=True) # B,C
  relevance = torch.einsum("bnc,bc->bn", x_norm, y_norm)
       # B,T*N
  relevance = relevance.reshape(B,T,N)
  return relevance
```