OST: Refining Text Knowledge with Optimal Spatio-Temporal Descriptor for General Video Recognition

Supplementary Material

6. Overview of Supplementary Material

In the supplementary material, we provide additional details in the following sections:

- Section 7: Further Analysis and Experiments
- Section 8: Details of Optimal Descriptor Solver
- Section 9: Dataset and Implementation Details
- Section 10: Demonstration of Prompts and Descriptors
- Section 11: Broader Impact and Limitation

7. Further Analysis and Experiments

7.1. Visualizations of Adaptive Transport Plan

We analyze the adaptive transport plan in our proposed *OD Solver*. Qualitative visualizations of the transport plan are illustrated in Fig. 6 and Fig. 7, with detailed explanations provided in the captions. We find that our proposed *OD Solver* can adaptly assign each descriptor to the video instance.

7.2. Visualizations of Attention Maps

We provide additional visualizations of the attention maps of our proposed **OST** in Fig. 8.

7.3. The Robustness of OST

We present case studies to illustrate the robustness of our proposed **OST**, specifically focusing on the transport plan depicted in Fig. 11 for scenarios where certain action steps are missing, and the attention maps in Fig. 12 where our **OST** effectively resolves category mismatches. Detailed analysis is provided within the captions of these figures.

7.4. Variant of Global Similarity

Besides the global similarity score computation illustrated in Eq. 9 in the main paper, an alternative global similarity score can be computed by initially determining the similarity between video representations and descriptor-level embeddings separately, and subsequently averaging these scores to derive the overall global video-descriptor similarity score. Although this approach may appear mathematically analogous to Eq. 9, the modified gradient flow during the training process could yield divergent outcomes. As demonstrated in Table 6, this implementation still exhibits sub-optimal performance in comparison to **OST**, thereby underscoring the superiority of our proposed method.

Table 6. Study on variants of global similarity score

Method	HMDB-51	UCF-101	K600
Variant 1	53.3	76.6	69.3
Variant 2	52.0	76.4	69.3
OST	**54.5**	**77.9**	**72.3**

8. Details of Optimal Descriptor Solver

8.1. Theoretical Analysis

In this section, we will provide the theoretical analysis of the existence and unicity of the optimal transport plan $\boldsymbol{P}^*$ in our proposed *OD Solver*.

As discussed in Eq. 2 in the main paper, after obtaining a set of frame-level features $\boldsymbol{V} \in \mathbb{R}^{T\times d}$ and descriptor-level embedding for each class $\boldsymbol{D_k^s} \in \mathbb{R}^{N_s\times d}$, $\boldsymbol{D_k^t} \in \mathbb{R}^{N_t\times d}$. The cost matrix for each class can be defined as:

$$\boldsymbol{C_k^s} = 1 - cos(\boldsymbol{V}, \boldsymbol{D_k^s}), \quad \boldsymbol{C_k^t} = 1 - cos(\boldsymbol{V}, \boldsymbol{D_k^t}). \tag{16}$$

We can define the OT problem in Kantorovich formulation as:

$$\begin{aligned} \boldsymbol{P}^* &= \underset{\boldsymbol{P}\in\mathbb{R}^{T\times N}}{\arg\min} \sum_{i=1}^{T}\sum_{j=1}^{N} \boldsymbol{P}_{ij}\boldsymbol{C}_{ij} \\ \text{s.t.} \quad & \boldsymbol{P}\boldsymbol{e} = \boldsymbol{\mu}, \quad \boldsymbol{P}^\top \boldsymbol{e} = \boldsymbol{\nu}. \end{aligned} \tag{17}$$

However, solving the problem in Eq. 17 costs $O(n^3 logn)$-complexity, which is time-consuming. By adopting Sinkhorn [15] algorithm, we can define the entropy-regularized OT problem as:

$$\begin{aligned} \boldsymbol{P}^* &= \underset{\boldsymbol{P}\in\mathbb{R}^{T\times N}}{\arg\min} \sum_{i=1}^{T}\sum_{j=1}^{N} \boldsymbol{P}_{ij}\boldsymbol{C}_{ij} - \lambda \boldsymbol{H}(\boldsymbol{P}) \\ \text{s.t.} \quad & \boldsymbol{P}\boldsymbol{e} = \boldsymbol{\mu}, \quad \boldsymbol{P}^\top \boldsymbol{e} = \boldsymbol{\nu}. \end{aligned} \tag{18}$$

Adding an entropy regularization to the original OT problem makes the optimal regularized transport plan more straightforward. This allows us to calculate the optimal transport distance via Matrix Scaling Algorithms [49].

Lemma 1. *For $\lambda > 0$, the optimal transport plan $\boldsymbol{P}^*$ is unique and has the form $\boldsymbol{P}^* = diag(\boldsymbol{a})\boldsymbol{K}diag(\boldsymbol{b})$, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are two probability vectors of $\mathbb{R}^d$ uniquely*

defined up to a multiplicative factor and $\boldsymbol{K} = exp(-\boldsymbol{C}/\lambda)$.

Proof. The existence and unicity of $\boldsymbol{P}^*$ follows from the boundedness of $\boldsymbol{\mu}, \boldsymbol{\nu}$ and the strict convexity of minus the entropy. Consider $\mathcal{L}(P, \alpha, \beta)$ as the Lagrangian of Eq. 18, where α, β serve as the dual variables corresponding to the equality constraints in $\boldsymbol{\mu}, \boldsymbol{\nu}$:

$$\mathcal{L}(\boldsymbol{P}, \alpha, \beta) = \sum_{ij} \left(\frac{1}{\lambda} p_{ij} \log p_{ij} + p_{ij} m_{ij} \right) + \alpha^\top (\boldsymbol{P}\boldsymbol{e} - \boldsymbol{\mu}) + \beta^\top (\boldsymbol{P}^\top \boldsymbol{e} - \boldsymbol{\nu}). \tag{19}$$

For any couple (i, j), if $(\partial \mathcal{L}/\partial p_{ij} = 0)$, then it follows that $p_{ij} = e^{-1/2-\lambda_{\alpha_i}} e^{-\lambda m_{ij}} e^{-1/2-\lambda_{\beta_j}}$. Given that all entries in matrix $\boldsymbol{K}$ are strictly positive, we know from Sinkhorn's work [49] that there is a one-of-a-kind matrix in the form of $diag(\boldsymbol{a})\boldsymbol{K}diag(\boldsymbol{b})$ which fits the constraints given by $\boldsymbol{\mu}, \boldsymbol{\nu}$. Therefore, this matrix is necessarily $\boldsymbol{P}^*$, and we can calculate it using the Sinkhorn fixed point iteration:

$$\boldsymbol{a} \leftarrow \boldsymbol{\mu}/\boldsymbol{K}\boldsymbol{b}, \quad \boldsymbol{b} \leftarrow \boldsymbol{\nu}/\boldsymbol{K}^\top \boldsymbol{a}. \tag{20}$$

□

8.2. Pseudo-Code on OD Solver

As explained in the paper, our *OD Solver* is effective and simple to implement. In Algorithm 1, we show the PyTorch style pseudo-code on the implementation of our proposed *Optimal Descriptor Solver*.

9. Implementation Details

9.1. Dataset Details

We provide 6 video benchmarks used in our empirical studies:

Kinetic-400 [7] is a large-scale video dataset consisting of 10-second video clips collected from YouTube. 240,000 training videos and 20,000 validation videos in 400 different action categories.

Kinetic-600 [8] is an extension of Kinetics-400, consisting of approximarely 480,000 videos from 600 action categories. The videos are divided into 390,000 for training, 30,000 for validation, and 60,000 for testing. We mainly use its validation set for zero-shot evaluation.

UCF-101 [50] is a video recognition dataset for realistic actions, collected from YouTube, including 13,320 video clips with 101 action categories in total. There are three splits of the training and testing data.

HMDB-51 [31] is a relatively small video dataset compared to Kinetics and UCF-101. It has around 7,000 videos with 51 classes. HMDB-51 has three splits of the training and testing data.

Something-Something V2 [20] is a challenging temporal-heavy dataset which contains 220,000 video clips across 174 fine-grained classes.

ActivityNet [6] We use the ActivityNet-v1.3 in our experiments. ActivityNet is a large-scale untrimmed video benchmark, containing 19,994 untrimmed videos of 5 to 10 minutes from 200 activity categories.

9.2. Implementation Details

Zero-shot Experiments. We mainly follow the zero-shot setting in [41, 47]. We tune both the visual and textual encoder of a CLIP ViT-B/16 with 32 frames on Kinetics-400 for 10 epochs. The batch size is set as 256 and single-view inference is adopted during validation. We set the hyperparameters in the Sinkhorn algorithm [15] as $\lambda = 0.1$. We adopt the AdamW optimizer paired with a 8×10^{-6} initial learning rate with the CosineAnnealing learning rate schedule. Following [24, 34, 59], we perform a linear weight-space ensembling between the original CLIP model and the finetuned model with a ratio of 0.2.

We apply the following evaluation protocols in our zero-shot experiments: For UCF-101 and HMDB-51, the prediction is conducted on three official splits of the test data. We report average Top-1 accuracy and standard deviation. For Kinetics-600, following [11], the 220 new categories outside Kinetics-400 are used for evaluation. We use the three splits provided by [11] and sample 160 categories for evaluation from the 220 categories in Kinetics-600 for each split. We report average Top-1 and Top-5 accuracy and standard deviation.

Few-shot Experiments. For the few-shot setting, we utilize CLIP ViT-B/16 as We adopt the few-shot split from [41, 47] that randomly samples 2, 4, 8, and 16 videos from each class on UCF-101, HMDB-51, and Something-Something V2 for constructing the training set. For evaluation, we use the first split of the test set on UCF-101, HMDB-51, and Something-Something V2. We utilize 32 frames during training and validation. Top-1 accuracy with single-view inference is reported. We set the batch size as 64 and train for 50 epochs in few-shot experiments.

Fully-supervised Experiments. For fully-supervised studies, we base our approach on Text4Vis [60] to conduct experiments in frozen text settings and keep the hyperparameters and data augmentations consistent with the baseline. We vary CLIP ViT-B/32, and ViT-B/16 as encoder and train with 8, and 16 frames, respectively. We report Top-1 accuracy using single-view inference.

Data Augmentation Recipe. For a fair comparison, we largely follow the data augmentations in ViFi-CLIP [47] for zero-shot and few-shot experiments and follow the recipe in Text4Vis [60] for fully-supervised experiments. The details for our data augmentation recipe are shown in Table 7.

Training and Testing. We employ the identical alignment

Algorithm 1 PyTorch style pseudo-code on Optimal Descriptor Solver

```
def OptimalDescriptorSolver(video_emb, descriptor_emb):
    A, N, D = descriptor_emb.shape # Get the shape of descriptor embeddings
    B, T, D = video_emb.shape # Get the shape of video embeddings
    sim = torch.einsum('b t d, a n d->t n b a', video_emb, descriptor_emb)  # Compute the similarity
    sim = rearrange(sim, 't n b a->(b a)t n')  # Rearrange dimensions
    cost_mat = 1 - sim  # Calculate the cost matrix
    pp_x = torch.zeros(B*A, T).fill_(1. / T)  # Initialize the horizontal probability vector
    pp_y = torch.zeros(B*A, N).fill_(1. / N)  # Initialize the vertical probability vector
    with torch.no_grad():
        KK = torch.exp( - cost_mat / eps)  # Calculate the cost matrix with exponentiation
        P = Sinkhorn(KK, pp_x, pp_y)  # Apply Sinkhorn algorithm to obtain the optimal transport plan P

    # Using optimal transport plan P to obtain logits
    score_ot = torch.sum(P * sim, dim=(1, 2))  # Frobenius inner product
    logits = score_ot.view(B, A)  # Classification logits
    return logits

def Sinkhorn(K, u, v):
    r = torch.ones_like(u)  # Initialize r as a tensor of ones with the same shape as u
    c = torch.ones_like(v)  # Initialize c as a tensor of ones with the same shape as v
    thresh = 1e-2  # Threshold to determine convergence in Sinkhorn iterations
    max_iter = 100 # Maximum number of Sinkhorn iterations
    # Sinkhorn iteration
    for i in range(max_iter):  # Iterate up to the maximum number of iterations
        r0 = r # Save the previous iteration's r
        r = u / torch.matmul(K, c.unsqueeze(-1)).squeeze(-1)  # Update r
        c = v / torch.matmul(K.permute(0, 2, 1), r.unsqueeze(-1)).squeeze(-1)  # Update c
        err = (r - r0).abs().mean()  # Calculate the mean absolute change in iterations
        if err.item() < thresh:  # If the change is below the threshold, stop iterating
            break
        P = torch.matmul(r.unsqueeze(-1), c.unsqueeze(-2)) * K  # Obtain the final transport plan P
    return P

```

Table 7. Data augmentation recipe for video recognition.

Setting	Zero/Few-shot	Fully-supervised
Augmentation		
RandomFlip	0.5	0.5
Crop	*MultiScaleCrop*	*RandomSizedCrop*
ColorJitter	0.8	0
GrayScale	0.2	0.2
Label smoothing	0	0
Mixup	0	0
Cutmix	0	0

mechanism throughout both the training and testing phases. The only difference lies in the application of contrastive-style operations during training, where logits are obtained exclusively from descriptors within the current mini-batch. During testing, classification scores are calculated against descriptors from all classes.

10. Demonstration of Prompts and Descriptors

10.1. Prompting the Language Model

We provide our prompts for generating *Spatio-Temporal Descriptors* in Fig. 9 and Fig. 10, respectively. We provide details in the figure captions.

10.2. Additional Examples of Spatio-Temporal Descriptors

In this section, we provide additional examples of the *Spatio-Temporal Descriptors*.

Descriptors for action category "Adjusting Glasses":

Spatio Descriptor:

1. person wearing glasses
2. hand adjusting glasses
3. glasses sliding on face
4. fingers pushing up glasses

Temporal Descriptor:

1. Push the glasses up the bridge of your nose
2. Align the temples with your ears
3. Adjust the nose pads for comfort
4. Ensure that the glasses rest comfortably on your face

Descriptors for action category "Assembling Bicycle":

Spatio Descriptor:

1. Bicycle frame
2. Handlebars
3. Wheels
4. Pedals

Temporal Descriptor:

1. Attach the front wheel to the bicycle frame using a wrench and follow the specified torque setting.

2. Secure the handlebars onto the front fork by tightening the stem bolts with an Allen wrench.
3. Install the pedals onto the crank arms by screwing them in clockwise.
4. Adjust the seat height to the desired position and tighten the seat clamp to secure it.

Descriptors for action category "Building Sandcastle":

Spatio Descriptor:
1. beach
2. sand
3. castle
4. bucket

Temporal Descriptor:
1. Dig a shallow hole in the sand for the base
2. Fill the hole with wet sand and pack it down firmly
3. Create a large mound of sand on top of the base
4. Use your hands or tools to shape the sand into walls and towers

Descriptors for action category "Opening Wine Bottle":

Spatio Descriptor:
1. wine bottle
2. corkscrew
3. uncorking
4. pouring

Temporal Descriptor:
1. Hold the wine bottle firmly
2. Remove the foil or plastic covering from the top of the bottle
3. Insert the corkscrew into the center of the cork
4. Twist the corkscrew counterclockwise to remove the cork

Descriptors for action category "Planing Wood":

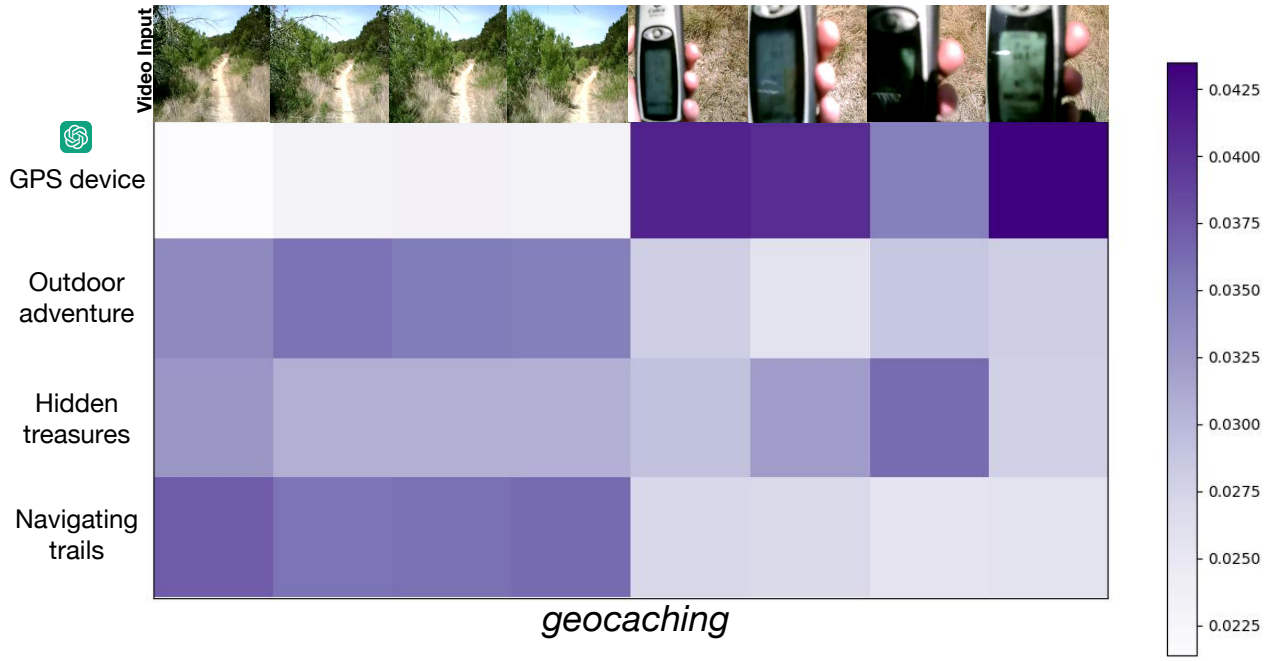
Spatio Descriptor:
1. wood
2. sawdust
3. saw
4. workbench

Temporal Descriptor:
1. Measure and mark the dimensions of the wood piece
2. Cut the wood according to the marked measurements
3. Smooth the edges of the cut wood using sandpaper
4. Apply a coat of varnish or paint to protect and enhance the appearance of the wood

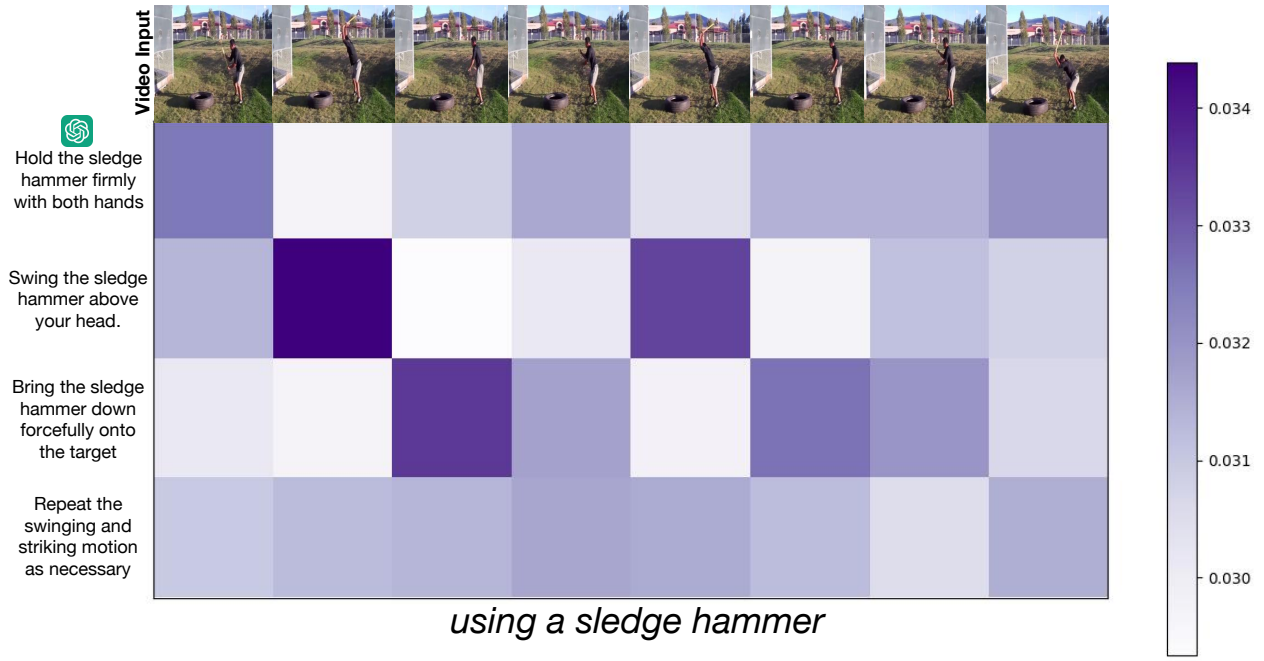
11. Broader Impact and Limitation

OST represents an effective way to utilize external knowledge to adapt pre-trained visual-language models for general video recognition. Our approach can benefit zero-shot, few-shot, and fully-supervised video recognition with no modification to the model architecture and minor additional computational costs. Furthermore, the proposed *Spatio-Temporal Descriptor* can greatly reduce the semantic similarity of action categories. **The employment of LLMs to generate corresponding descriptors can be readily extended to various unseen action categories, allowing the open-vocabulary understanding of actions in the wild.**

However, the quality of descriptors directly connects to the final performance. The process of generating descriptors highly depends on the knowledge learned by the LLM, which is only partially controllable by varying the prompts. Additionally, our findings suggest that the informational needs for describing actions differ across various categories. Relying solely on four Spatio-Temporal Descriptors might not be ideal for every category. An adaptive approach, where the number of descriptors is tailored to each category, would likely be more effective.

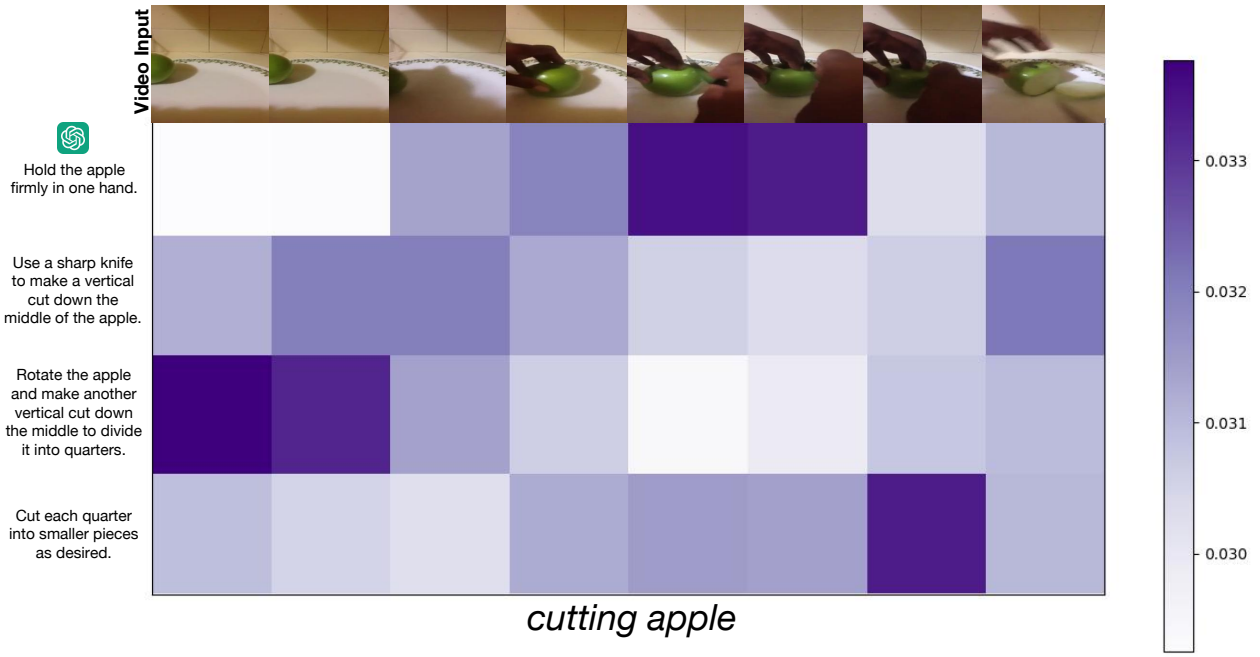


(a) Adaptive transport plan of action category "*geocaching*"

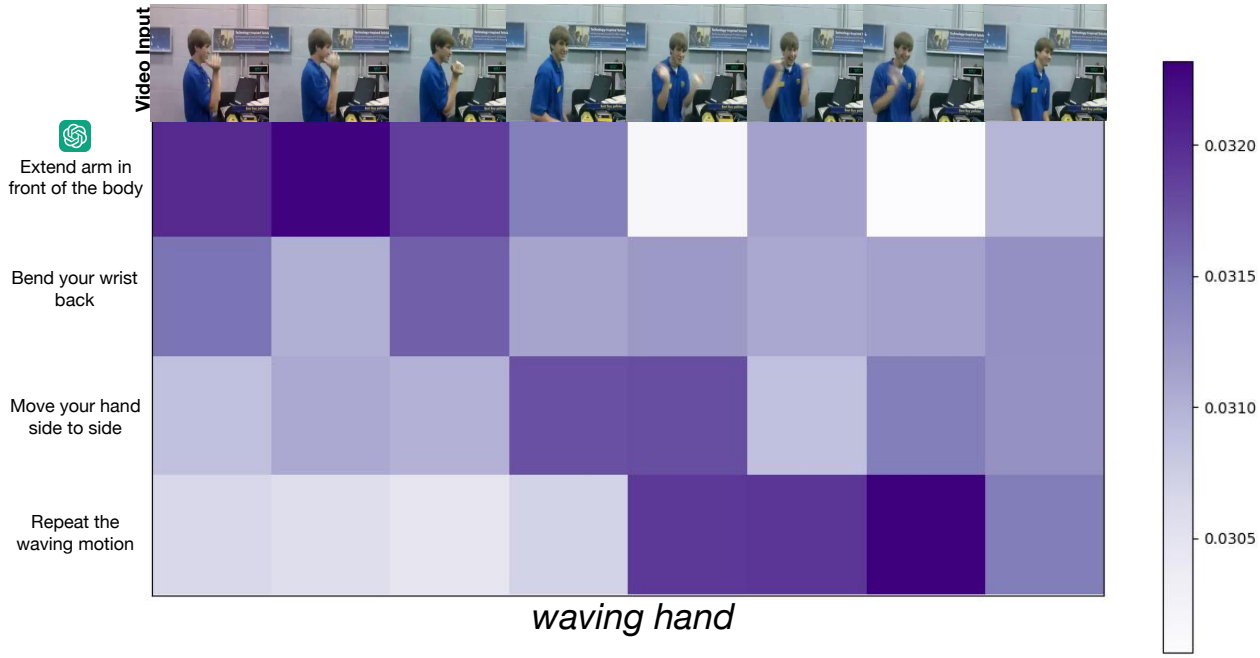


(b) Adaptive transport plan of action category "*using a sledge hammer*"

Figure 6. Visualization of the adaptive transport plan. Our *OD Solver* not only integrates various visual cues—such as *GPS devices*, *navigation trails* in Fig. 6a, and *hammer-swinging motions* in Fig. 6b, but also greatly reduce the detrimental effects of the noisy descriptors that often arise from the hallucination issues associated with LLMs, such as misleading '*hidden treasures*' in Fig. 6a or '*repeat the swinging*' in Fig. 6b). It is important to note that while the absolute variances among transport plans are relatively small, their substantial relative differences are critical in optimal matching.



(a) Adaptive transport plan of action category "*cutting apple*"



(b) Adaptive transport plan of action category "*waving hand*"

Figure 7. Visualization of the adaptive transport plan. Our investigation reveals that our *OD Solver* can synchronize different action steps described by *Temporal Descriptor* with corresponding video sequences. For example, it accurately coordinates actions such as '*holding an apple firmly in one hand*' in Fig. 7a, '*extend the arm in front of the body*', and '*moving the hand from side to side*' in Fig. 7b with the help of corresponding *Temporal Descriptors*.

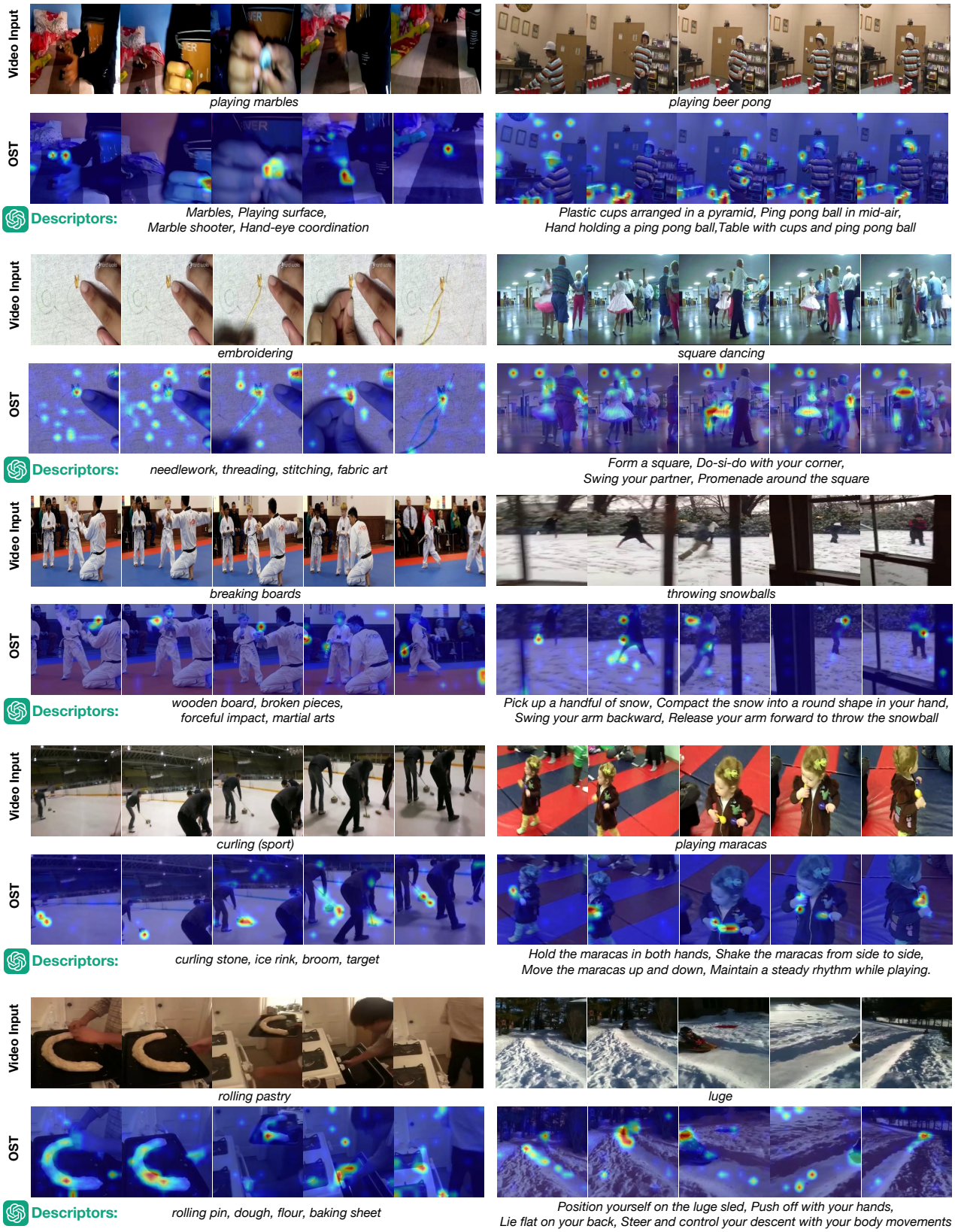


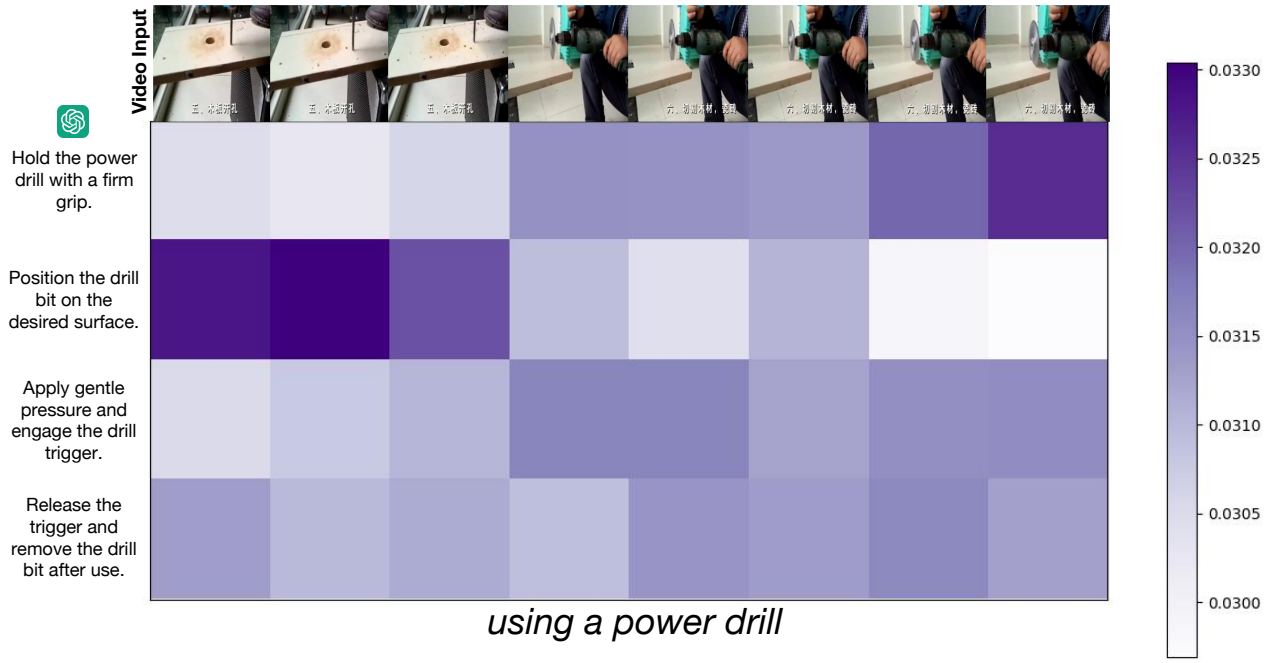
Figure 8. Additional visualizations of attention maps. The attention maps corresponding to the *Spatio Descriptors* and *Temporal Descriptors* are depicted on the left and right sides, respectively. The visualizations reveal that our proposed **OST** consistently focuses on specific static objects and temporal salient parts. This consistent focus underscores the efficacy of our approach.

```
openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
    {
      "role": "system",
      "content":
        "You are an intelligent chatbot designed for providing meaningful class-label augmentations for
        classification tasks. "
        "Your task is to give corresponding meaningful and distinguishable text  descriptions. Here's
        how you can accomplish the task:"
        "------"
        "##INSTRUCTIONS: "
        "- Focus on the static visual cues that may benefit visual-side classification.\n"
        "- Give the key descriptors that can be found within a single image.\n"
        "- Try to focus on object-level cues, such as obvious objects or scenes that may include in the
         image.\n"
        "- Do not include descriptor that only contains 'person'"
    },
    {
      "role": "user",
      "content":
        f"Please give me a long list of descriptors for action: {category}"
        f"Provide your answer only as the description it self, {num_captions} descriptors in total."
        "Please generate the response in the form of a Python list string that consists of the
        descriptors you provide."
        "DO NOT PROVIDE ANY OTHER OUTPUT TEXT OR EXPLANATION. Only provide the Python
        list string. "
        "For example, your response should look like this: [\"descriptor1\", \"descriptor2\" ... ]."
    }
  ]
)
```

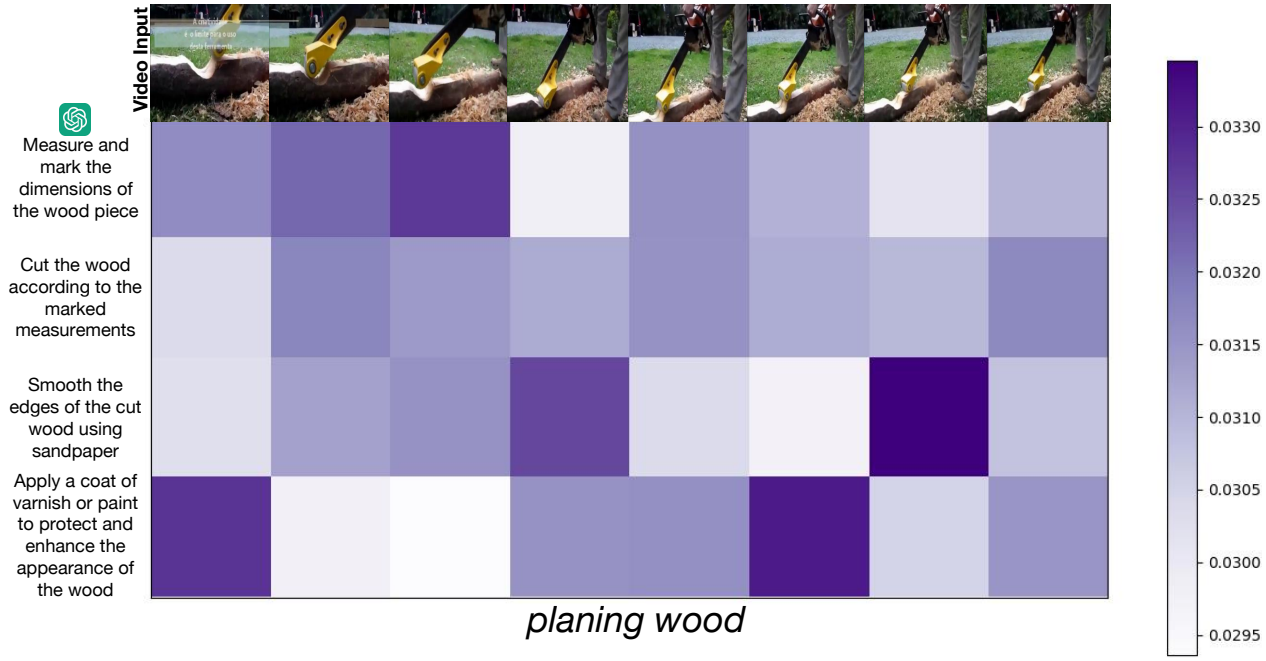
Figure 9. Prompt for generating *Spatio Descriptors*. The generated *Spatio Descriptors* are intended to capture static visual elements that can be discerned from a single image, such as environments and objects. So we prompt the LLM to prioritize and interpret object-level cues.

```
openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
    {
      "role": "system",
      "content":
        "You are an intelligent chatbot designed for providing meaningful class-label augmentations for
        classification tasks. "
        "Your task is to give corresponding meaningful and distinguishable text descriptions. Here's
        how you can accomplish the task:"
        "------"
        "##INSTRUCTIONS: "
        "- You can try to include as much verb as possible.\n"
        "- Do not include descriptor that only contains 'person'"
    },
    {
      "role": "user",
      "content":
        f"Please give me a long list of decompositions of steps for action: {category}"
        f"Provide your answer only as the steps it self, {num_captions} steps in total."
        "Please generate the response in the form of a Python list string that consists of the steps you
        provide."
        "DO NOT PROVIDE ANY OTHER OUTPUT TEXT OR EXPLANATION. Only provide the Python
        list string. "
        "For example, your response should look like this: [\"step1\", \"step2\" ... ]."
    }
  ]
)
```

Figure 10. Prompt for generating *Temporal Descriptors*. For *Temporal Descriptors*, our aim is to decompose the action classes in a step-by-step manner, detailing how an action progresses over time. To enhance the adapted model's capacity to learn action verbs during the training phase, we prompt the LLM to include a comprehensive range of verbs.



(a) Adaptive transport plan of action category "*using a power drill*"



(b) Adaptive transport plan of action category "*planing wood*"

Figure 11. Visualization of cases where certain steps are missing. Our study demonstrates the efficacy of our proposed *OD Solver* in accurately identifying instances when specific action steps are either missing or altered. As depicted in Fig. 11a, the actions '*Engage the drill trigger*' and '*Release the trigger*' are absent from the video sequence. With the help of our proposed *OD Solver*, our model is capable of adaptively aligning the video instance with its corresponding category descriptor, effectively compensating for these absences. This capability is further evidenced in Fig.11b, where the action '*Cut the wood according to the marked measurements*' is missing from the video instance. Our *OD Solver* adeptly adjusts to the modified sequence by assigning lower weights to the descriptors associated with the missing actions, demonstrating the method's robustness in handling incomplete or altered action sequences.

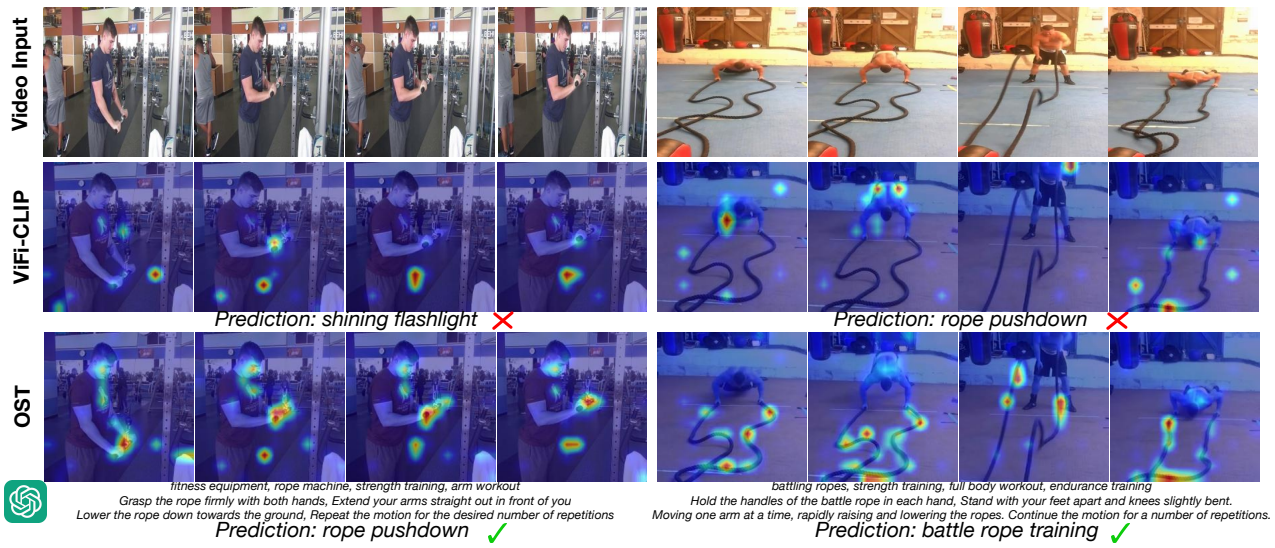


Figure 12. Visualization of cases where our proposed *Spatio-Temporal Descriptors* successfully resolves category mismatch. Relying solely on the category names, ViFi-CLIP misidentifies the equipment as a '*flashlight*' and misinterprets '*rope pushdown*'. In contrast, aided by *Spatio-Temporal Descriptors*, our **OST** accurately discerns the action, with a particular focus on temporally significant elements such as the man's hand and the rope.