# A Robustness to Domain Shift: Zero-shot Cross-Domain Transfer

Table 4: Evaluating robustness to domain shift. We train the models on SSv2-label and perform zero-shot action classification on out-of-domain datasets, i.e., Moments-In-Time [37] and Temporal-Kinetic [44]. $\Delta$ indicates the relative increase/decrease compared to the backbone.

| **Method** [Patcher Training Loss] | **Zero-shot Cross-domain Transfer** | | | |
|---|---|---|---|---|
| | **Moments-In-Time** | | **Temporal-Kinetic** | |
| | Val (Acc) | $\Delta(\%)$ | Val (Acc) | $\Delta(\%)$ |
| InternVideo Backbone | 23.3 | - | 57.7 | - |
| KP-Transformer FT [VTC] | 16.5 | -29% | 44.7 | -23% |
| KP-Perceiver FT [VTC] | 9.9 | -58% | 24.7 | -57% |
| Side-Tuning [60] [VTC+DVDM] | 21.2 | -10% | 54.5 | -6% |
| **PAXION** [VTC+DVDM] | 21.6 | -7% | 49.7 | -14% |
| w/o Knowledge Fuser | 4.3 | -82% | 16.3 | -72% |
| w/ Backbone Ensemble | **23.9** | **+3%** | **58.1** | **+1%** |

Humans acquire action knowledge through multisensory interactions, and have the remarkable ability to generalize to new objects and scenarios. Similarly, our ultimate goal is to learn the underlying rules of action knowledge that is generalizable to unseen domains. However, it is highly challenging when we are given only domain-specific datasets. For instance, the SSv2 dataset [12] only has 174 action classes, which is insufficient to capture the full range of open-world actions. The Ego4d dataset is limited to ego-centric videos, making it difficult to generalize to other types of videos. Training on such domain-specific data can easily lead to overfitting to spurious features and introduce catastrophic forgetting of tasks from other domains. In this section, we further explore *whether* PAXION *is robust to domain shift* and *whether the learned action knowledge can bring positive transfer to action-centric tasks on unseen domains*.

We consider a zero-shot cross-domain transfer setting where we directly apply the models trained on SSv2-label [23] to unseen domains. We consider two zero-shot action classification tasks based on **Moments-In-Time** [37][3] and **Temporal-Kinetic** [44]. Moments-In-Time contains 305 action classes with diverse types of videos that are distinct from SSv2, including movie clips, stock footages, and cartoons. Temporal-Kinetic contains 32 manually selected action classes from Kinetic-400, with a special focus on temporal reasoning. We directly use the action labels (e.g., *"bouncing"* and *"kicking"*), as the text candidates for the zero-shot classification [42], which introduces additional domain shifts in terms of text distribution compared with the annotations in SSv2-label (e.g., *"book falling like a rock"*).

**Fusing with the backbone improves robustness to domain shift.** Table 4 shows the zero-shot action classification accuracy and the relative difference $\Delta(\%)$ compared with the frozen backbone. We find that adding the Knowledge Fuser effectively increases robustness to domain shift, as reflected by a smaller negative $\Delta$. The Side-tuning also demonstrate similar benefit via alpha blending between the Knowledge Patcher and the backbone.

**Positive transfer can be achieved by ensembling the Knowledge Fuser (KF) with the backbone.** We further propose a simple inference trick, **Backbone Ensemble**, which combines the output probability from the KF and the backbone model through addition. Specifically, the final prediction of the action class index $c \in 0, 1, ..., C$ is computed as $c = \arg\max_{i \in 0,1,...,C}\,(p_a(i = c) + p_b(i = c))$, where $C$ is the number of classes, $p_a$ and $p_b$ are the predicted probability distribution from the KF and the backbone respectively. We obtain the final prediction by ranking the combined probability of the action text candidates. Our experiments show that this simple inference technique can effectively enhance zero-shot performance and achieve positive transfer on unseen domains.

# B Details of Action Dynamics Benchmark (ActionBench)

We construct ActionBench based on two existing video-language datasets with fine-grained action text annotation, Ego4d [13] and SSv2 [12]. To automatically generate the antonym text for the Action

[3] We subsample 2k instances for doing this evaluation.

Table 5: ActionBench Statistics

| **Dataset** | **#Train** | **#Eval** | **Video Type** |
|---|---|---|---|
| ActionBench-Ego4d | 274,946 | 34,369 | first-person |
| ActionBench-SSv2 | 162,475 | 23,807 | first-person, third-person |

Antonym task, we leverage WordNet [35][4] to find antonyms for verb text tokens. Additionally, we construct an additional verb-to-antonym mapping by leveraging ChatGPT[5] and manual curation, since the WordNet database does not cover all verbs in the action taxonomy of the dataset. Furthermore, to ensure that the action antonym indeed forms a negative video-text pair with the original video, we exclude verbs that do not have a semantically reasonable antonym, such as "use" and "look". For Ego4d, we consider a subset of EgoClip [31] annotations, for SSv2 we consider the entire dataset. The final statistics of the training and evaluation splits can be found in Table 5. For SSv2, since the test set does not provide label annotation, i.e., annotation with filled object names, we report scores on the validation set. For Ego4d, we evaluate on the test set. For results in Table 1, we train the Knowledge Patcher variants for one epoch on the training sets and report the accuracy on the evaluation sets. We downsampled the videos into 224x224 in scale with a frame rate of 8 fps for both training and evaluation. For human evaluation, we randomly sample 50 instances for the Action Antonym and the Object Replacement task, and another 50 instances for the Video Reversal task. The human evaluation is done by the authors.

# C Identifying State-change Salient Videos for Action-Temporal Matching (ATM)

As detailed in § 3.1, we formulate the Action-Temporal Matching (ATM) loss as distinguishing reversed video from the original one given an action text. ATM requires the model to learn the correlation between the correct temporal ordering of the visual observations and the corresponding actions. However, some actions, such as "wiping" and "holding", are repetitive or continuous and may not result in visible state-changes across the frames in the video clip. This can introduce additional noise for the ATM loss when the reversed video is indistinguishable from the original one. To address this issue, we propose two metrics to identify state-change salient videos by leveraging image-language foundation models. We use pretrained BLIP [27] to compute (1) **frame-text semantic change** $\delta_{vt}$, which indicates how the frame-text alignment changes across the first half and second half of the video; (2) **frame-frame similarity** $\theta_{vv}$, which indicates how different the frames from the first half and second half of the video are.

$$\delta_{vt} = \left| \frac{1}{N/2} \left( \sum_{i \in [0, N/2)} S(\mathbf{v_i}, \mathbf{t}) - \sum_{j \in [N/2, N)} S(\mathbf{v_j}, \mathbf{t}) \right) \right| \tag{1}$$

$$\theta_{vv} = S\left( \frac{\sum_{i \in [0, N/2)} \mathbf{v_i}}{N/2}, \frac{\sum_{j \in [N/2, N)} \mathbf{v_j}}{N/2} \right) \tag{2}$$

where $N$ is the total number of sampled frames[6], $\mathbf{v}$ and $\mathbf{t}$ are the frame image embedding and the text embedding from pretrained BLIP encoders, $S$ denotes cosine similarity.

Intuitively, if we observe a large frame-text semantic change ($\delta_{vt}$) and a small frame-frame similarity ($\theta_{vv}$), we could expect to see salient state-changes between the first half and the second half frames. We empirically set a threshold for $\delta_{vt}$ and $\theta_{vv}$. During training, we only compute ATM loss on videos that satisfy $\delta_{vt} > 0.003$ and $\theta_{vv} < 0.95$. The metrics are computed off-line thus do not bring computational overhead during training. Figure 7 shows an example of the videos that are kept and skipped based on the computed metrics.

[4] We use the WordNet Interface from NLTK https://www.nltk.org/howto/wordnet.html.
[5] https://openai.com/blog/chatgpt.
[6] We use $N = 8$ in our experiments.

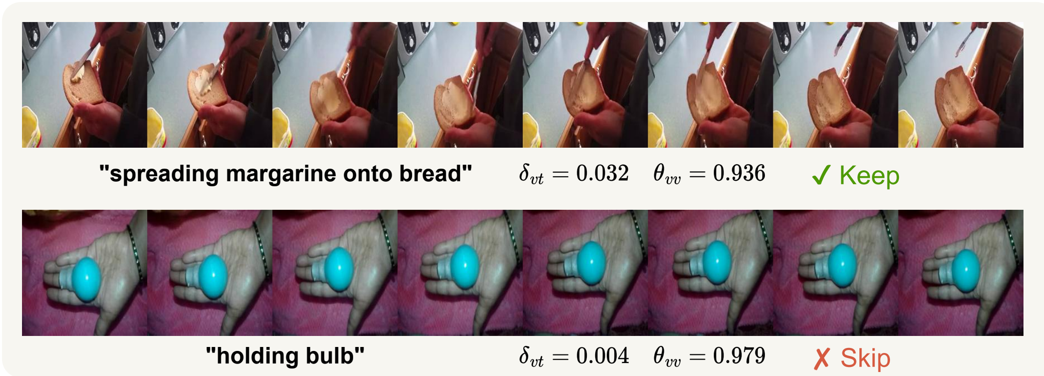


Figure 7: Example of identifying state-change saliency in videos for forward dynamics modeling. $\delta_{vt}$ and $\theta_{vv}$ indicates *frame-text semantic change* and *frame-frame similarity* metrics.

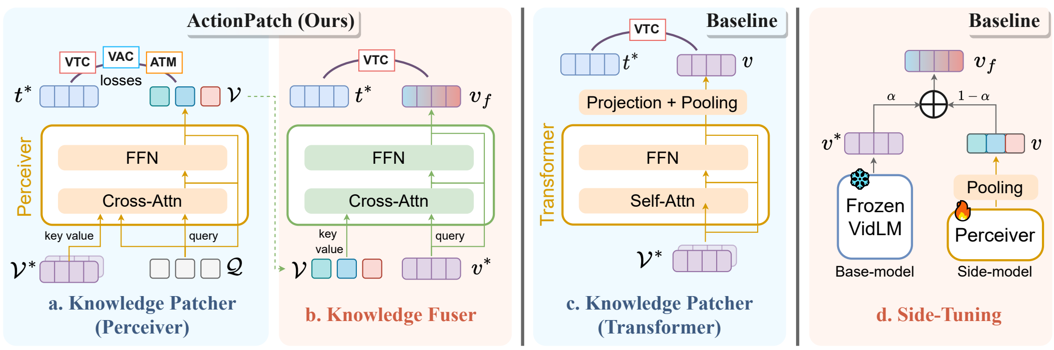


Figure 8: Detailed architecture of Knowledge Patcher (Perceiver), Knowledge Patcher (Transformer), Knowledge Fuser and Side-Tuning fuser.

# D Implementation Details

## D.1 Architecture Details.

Figure 8 shows detailed architecture of the Knowledge Patcher and Knowledge Fuser in our PAXION framework, as well as the baseline variants being compared in Tables 1, 2 and 3.

**Knowledge Patcher (Perceiver).** The Perceiver-based Knowledge Patcher contains a single cross-attention layer and a two-layer feedforward network. The Perceiver module performs cross-attention between a sequence of learnable latent queries $\mathcal{Q} \in \mathbb{R}^{l,d}$ and the raw visual embeddings $\mathcal{V}^* \in \mathbb{R}^{P,D}$ from the frozen backbone, where $P$ denotes the visual token length and $D$ represents the hidden dimension of the visual backbone. Since the user-defined sequence length $l$ and hidden dimension $d$ of the learnable latent queries are typically much smaller than $P$ and $D$ from the backbone, the Perceiver module serves as an information bottleneck that extracts knowledge-specific features from the raw visual features. For instance, in the case of InternVideo [51] backbone, we set $l = 16, d = 768$ which is much smaller than $P = 1576, D = 1024$ for each video clip with 8 sampled frames. Similar to BLIP-2 [26], when computing the similarity between the visual tokens $\mathcal{V} \in \mathbb{R}^{l,d}$ from the Knowledge Patcher and the single textual feature vector $t^* \in \mathbb{R}^d$, we first compute the pairwise similarity between each visual token and the text feature vector, and then take a maximum across all visual tokens as the final video-text similarity. The results in Table 1 demonstrate the Perceiver-based Knowledge Patcher achieves competitive or better performance compared to the Transformer variant while being 2-3 times smaller. Additionally, we measure the computation overhead of the two variants, and find that the Perceiver variant requires 10 times fewer *multiply-add operations* than the Transformer variant. This further demonstrate that Perceivers can serve as effective and efficient extractors for knowledge-specific features.

Table 6: Detailed configurations for methods in Tables 2 and 3, and Figure 6.

| Method | has Knowledge Fuser? | Trainable Param# | Patching Objectives | Fusing/Finetuning Objectives |
|---|---|---|---|---|
| KP-Transformer FT | ✗ | 8.4M (1.8%) | VTC | VTC |
| KP-Perceiver FT | ✗ | 4.2M (0.9%) | VTC | VTC |
| Side-Tuning | ✗ | 4.2M (0.9%) | VTC + DVDM | VTC |
| **PAXION** | ✓ | 8.2M (1.7%) | VTC + DVDM | VTC |
| KP+Finetune | ✗ | 4.2M (0.9%) | VTC + DVDM | VTC |
| KP[VTC]+KF | ✓ | 8.2M (1.7%) | VTC | VTC |

Table 7: Detailed training configurations for tasks in Tables 2, 3, and 4.

| Downstream Task | Patching Dataset | Patching #Epochs | Fusing/Finetuning Dataset | Fusing/Finetuning #Epochs |
|---|---|---|---|---|
| SSv2-label [23] | SSv2 | 1 | SSv2 | 1 |
| SSv2-template [23] | SSv2 | 1 | SSv2-template | 2 |
| Temporal-SSv2 [44] | SSv2 | 1 | SSv2-template | 2 |
| NExT-QA [53] | NExT-QA | 1 | NExT-QA | 4 |
| Moments-In-Time [37] | SSv2 | 1 | SSv2 | 1 |
| Temporal-Kinetic [44] | SSv2 | 1 | SSv2 | 1 |

**Knowledge Patcher (Transformer).** The Transformer variant of the Knowledge Patcher is a standard Transformer Encoder which contains a self-attention layer and a feedforward layer. The Transformer Encoder performs self-attention on the raw visual embeddings $\mathcal{V}^* \in \mathbb{R}^{P,D}$ from the frozen backbone and output an updated visual embedding $\mathcal{V} \in \mathbb{R}^{P,D}$. To obtain video-text similarity, we first project the visual embeddings into the same dimension as the textual feature vector $\boldsymbol{t}^* \in \mathbb{R}^d$ and then do mean pooling before computing dot product.

**Knowledge Fuser.** The Knowledge Fuser has the same architecture as the Knowledge Patcher which contains a single cross-attention layer and a two-layer feedforward network. In this case, we use the pooled visual feature from the backbone $\boldsymbol{v}^* \in \mathbb{R}^d$ to provide query and the Knowledge Patcher output $\mathcal{V} \in \mathbb{R}^{P,D}$ to provide key and value for the cross-attention. The intuition is to obtain a balanced representation for general downstream tasks by fusing the action-centric KP representation ($\mathcal{V}$) with the object-centric backbone representation.

**Side-Tuning.** As an alternative to the Knowledge Fuser, we consider Side-Tuning [60] for further integrating the Knowledge Patcher with the backbone. Side-Tuning contains a *base-model* and a *side-model*, where the base-model is pretrained and frozen and the side-model is trainable. In our setting, we treat the backbone as the base-model and initialize the side-model using the trained Knowledge Patcher. We then side-tune the Knowledge Patcher along with the backbone using alpha blending. Specifically, the final fused visual feature $\boldsymbol{v_f}$ is obtained by $\boldsymbol{v_f} = \alpha(\boldsymbol{v}^*) + (1-\alpha)\boldsymbol{v}$, where $\boldsymbol{v}^*$ is the mean-pooled backbone visual feature, and the $\boldsymbol{v}$ is the mean-pooled Knowledge Patcher feature. And $\alpha = Sigmoid(a) \in [0, 1]$, where $a$ a learnable scalar.

### D.2 Knowledge Patcher Training.

We use two Nvidia Tesla V100 (16GB) GPUs for all experiments. For the Knowledge Patcher variants in Table 1, we train them on the training set of the datasets in the ActionBench for one epoch with either VTC loss only or VTC + DVDM (VAC + ATM) loss. We use AdamW [33] optimizer with a learning rate of 1e-5 and a weight decay of 0.05. For the transformer variant, we use a batch size of 8 per GPU. For the Perceiver variant, we are able to increase the batch size to 32 per GPU due to the reduced computation complexity.

### D.3 Downstream Task Training.

Tables 6 and 7 shows detailed configurations for downstream task training with methods described in Tables 2 and 3, and Figure 6.

As shown in Table 7, the finetuning dataset for SSv2-label is identical to the SSv2 action knowledge patching dataset where the annotations are filled templates, such as “Book falling like a rock”. The SSv2-template dataset, on the other hand, contains the object-obscured version of the original SSv2 annotations such as “Something falling like a rock”. For the Video-to-Action Retrieval tasks, we consider two different subsets from the SSv2 validation set with the object-obfuscated annotations: SSv2-template [23] and Temporal-SSv2 [44]. SSv2-template contains all 174 action classes while Temporal-SSv2 contains 18 manually selected action classes that require more temporally-demanding distinctions, and cannot be distinguished using shuffled frames, such as “Approaching” and “Moving away”. In order to investigate the impact of the action knowledge patching, we do not finetune a dedicated model for the 18 action classes for Temporal-SSv2, but instead use the model trained on SSv2-template to directly evaluate on Temporal-SSv2. Therefore, when observed larger improvements on Temporal-SSv2, we can draw the conclusion that patching with action knowledge contributes more to action-centric tasks (§ 4.2).

The hyperparameters, such as the learning rate, are identical to those used during Knowledge Patching training. For Video-Text Retrieval (SSv2-label) and Video-to-Action Retrieval (SSv2-template, Temporal-SSv2), the DVDM (§ 3.1) objective includes VAC and ATM, while for Causal-Temporal VQA (NExT-QA), we only use VAC. This is because the training instances in NExT-QA are not formatted as video-text pairs but instead are in the format of multiple choice QA, making it not suitable for the ATM loss. Each video corresponds to one question and five candidate answers. We apply VAC to NExT-QA by adding action antonym text for each question as hard negative candidate answers.

For the downstream tasks (in Appendix A) for zero-shot cross-domain transfer (Moments-In-Time [37] and Temporal-Kinetic [44]), we use the model trained on SSv2 to perform zero-shot evaluation.

## E Additional Qualitative Analysis

Figures 9 and 10 show additional qualitative examples on downstream tasks. The examples in demonstrate that PAXION improves understanding of challenging actions that require fine-grained temporal reasoning on the frames. For example, whether it is `‘‘pretending’’` to do something or actually doing that, and whether an object is moving `‘‘towards’’` or `‘‘away’’` from the camera.

In Figure 11, we show failure cases of PAXION to discuss remaining challenges. We find that PAXION still struggle to understand *negation* and *spatial attributes*. For example, both VTC-Finetune baseline and PAXION fail to distinguish `‘‘without letting it drop down’’` from `‘‘then letting it drop down’’`. For questions that require fine-grained spatial information of objects such as `‘‘how many goats can be spotted’’`, PAXION cannot perform well. Potential solutions including incorporating the patched VidLM with a code language model to disentangle perception and reasoning similar to ViperGPT [46]. By leveraging the strong logical reasoning ability of a code language model, we can easily solve the negation and counting problems by creating code scripts with booleans and loops, and then use the VidLMs as "API calls".

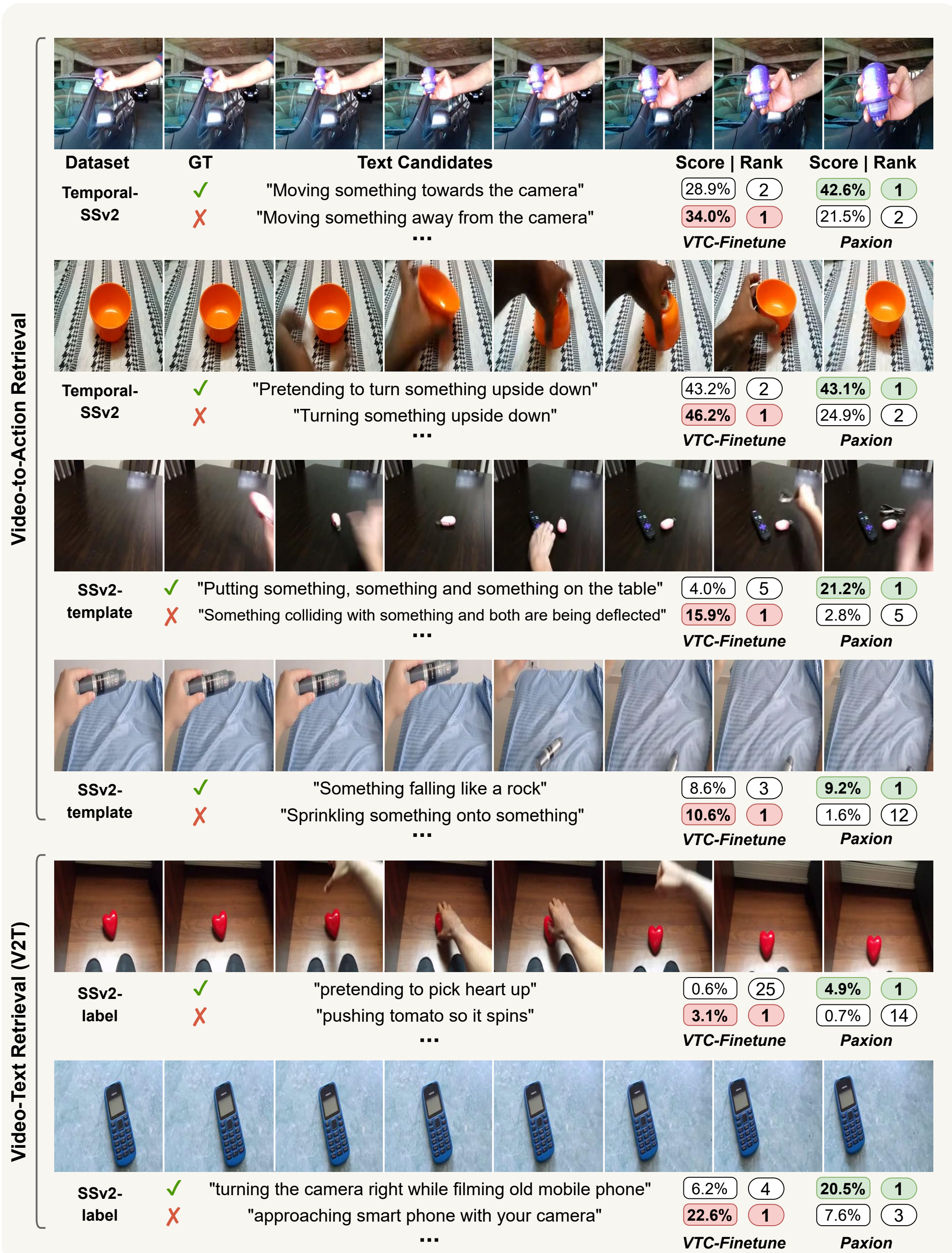


Figure 9: Additional qualitative examples (Retrieval).

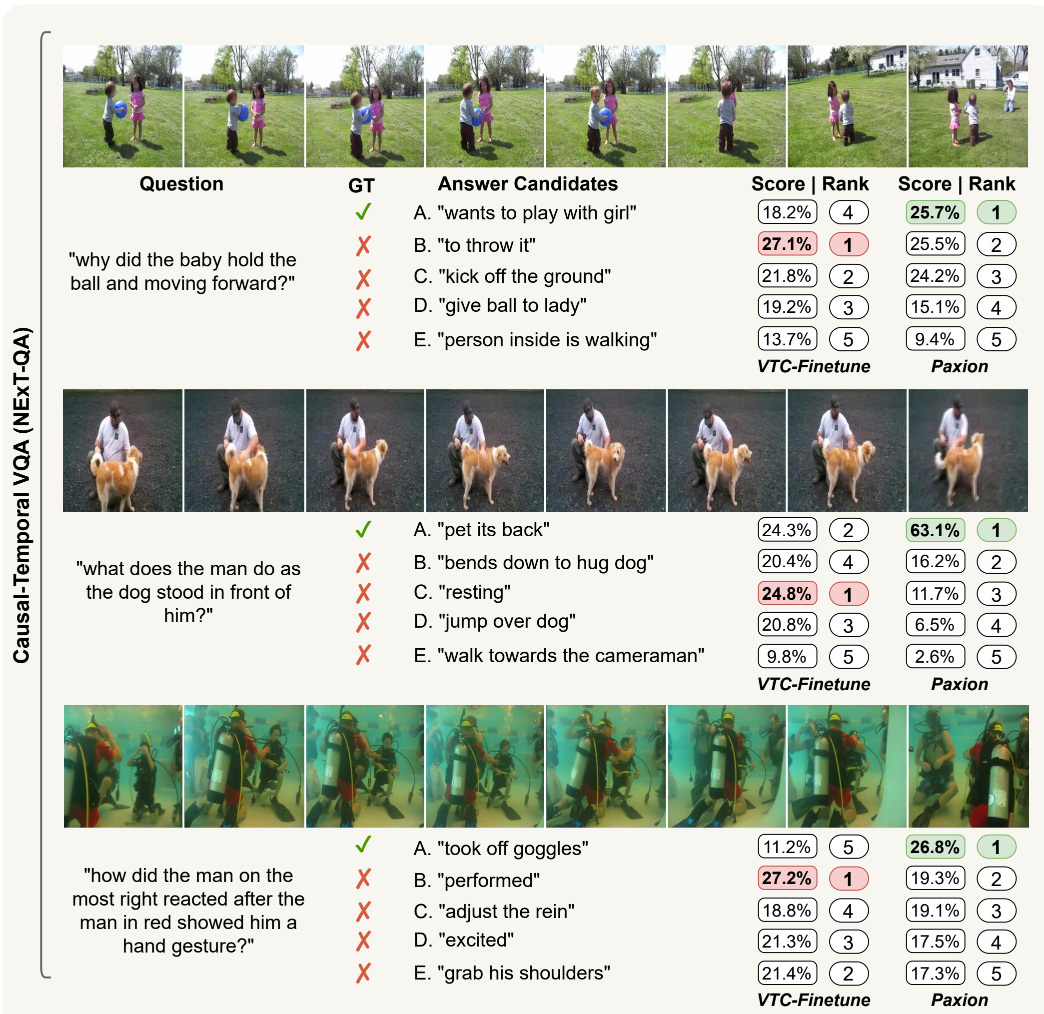


Figure 10: Additional qualitative examples (VQA).

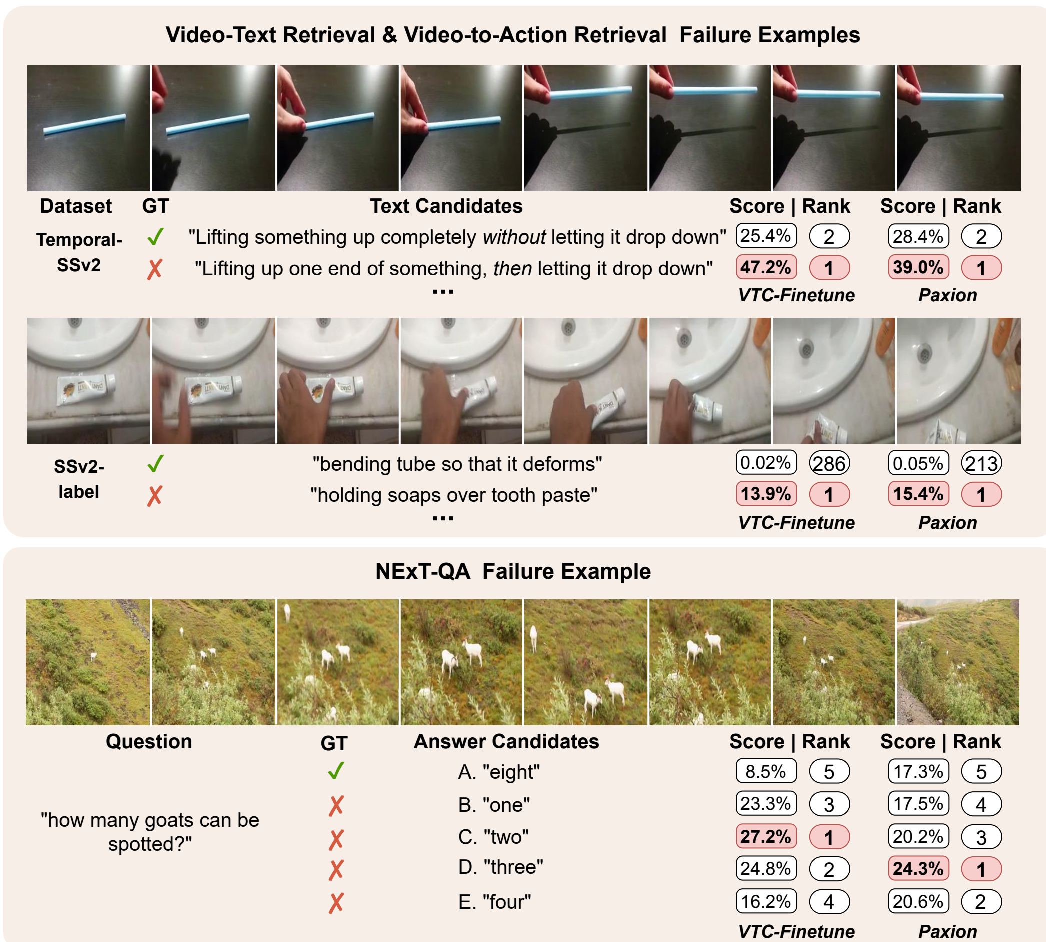


Figure 11: Failure examples.