APPENDIX

This appendix is organized as follows:

- **Appendix A** introduces more details about the implementation.
- **Appendix B** presents more experimental results and analysis.
- **Appendix C** provides more visualizations for demonstration.
- **Appendix D** shares further discussion on limitations and future directions.
- **Appendix E** clarifies the use of the large language model.

A IMPLEMENTATION DETAILS

A.1 DYNAMIC FUSION MODULE

As described in Section 3, we design a simple yet effective dynamic fusion module to bridge interactions among multi-scale frame features produced by the ActionFormer backbone, thereby refining temporal representations. The internal structure of this module is illustrated in Figure 5. Frame features at each level, with different temporal lengths, are first passed through a convolutional layer followed by a normalization layer to capture temporal information at varying granularities. For each feature, the features from its adjacent levels are up-sampled or down-sampled along the temporal dimension via linear interpolation to match its length, after which a weighted sum is applied. The weights are learnable and specific to each feature. The resulting features preserve their original length and dimensionality while being enriched with aggregated temporal information. An ablation study in Table 3e validates the effectiveness of this dynamic fusion module for mistake detection.

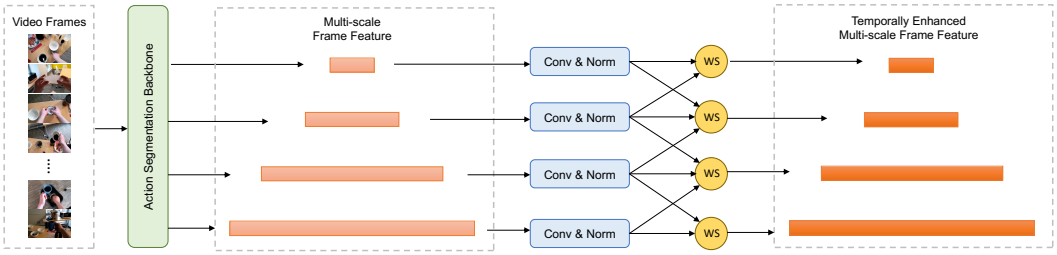

Figure 5: Structure of the dynamic fusion module. Each *Conv&Norm* block represents a convolutional layer followed by a normalization layer, and each *WS* block denotes a weighted-sum operation.

A.2 EFFECT-FRAME SAMPLING

In this work, we perform action-effect modeling by leveraging the effect knowledge embedded in effect frames. To sample effect frames within action segments, we first predict key objects that indicate the action effect and then apply semantic scoring to evaluate the relevance of each video frame to the effect. Figure 7a presents the complete GPT-4o prompt used for key-object prediction and generation of textual effect descriptions.

A.3 ACTION SCENE GRAPH

In this work, we construct a symbolic action scene graph from sampled effect frames to generate textual supervision for effect learning. This subsection details the implementation of the action scene graph, including: (1) generating the scene graph by prompting GPT-4o and parsing its output, (2) decomposing the graph into object-state and spatial-relation subgraphs, and (3) presenting a concrete example that illustrates the overall process.

A.3.1 SCENE GRAPH CONSTRUCTION

We prompt the powerful GPT-4o to recognize the action scene and output the scene information. Figure 7b shows the complete prompt we used. Scene information is returned in the format of the

dictionary (JSON file), and we design algorithms to parse the output and construct the action scene graph based on it. Algorithm 1 describes the graph construction process, where the input includes the object relation dictionary (*Relation_Dict*) and the object attribute dictionary (*Attribute_Dict*) generated by the VLM. The output consists of two dictionaries: one containing the graph nodes categorized by type (*Nodes*) and the other storing the edges connecting pairs of nodes (*Nodes*).

A.3.2 Action Scene Graph Decomposition

Algorithm 2 describes the graph decomposition process, where the input includes *Nodes* and *Nodes* obtained from graph construction, as well as the specified action-related objects generated by effect frame sampling. The outputs are separate sub-graphs of object state (*S-Graph*) and object spatial relationships (*R-Graph*).

A.3.3 Example of Scene Graph Processing

Figure 6 illustrates how an action scene graph is constructed and decomposed for the effect frame of *Pouring Water on Coffee Ground* within the task of *Making Coffee*.

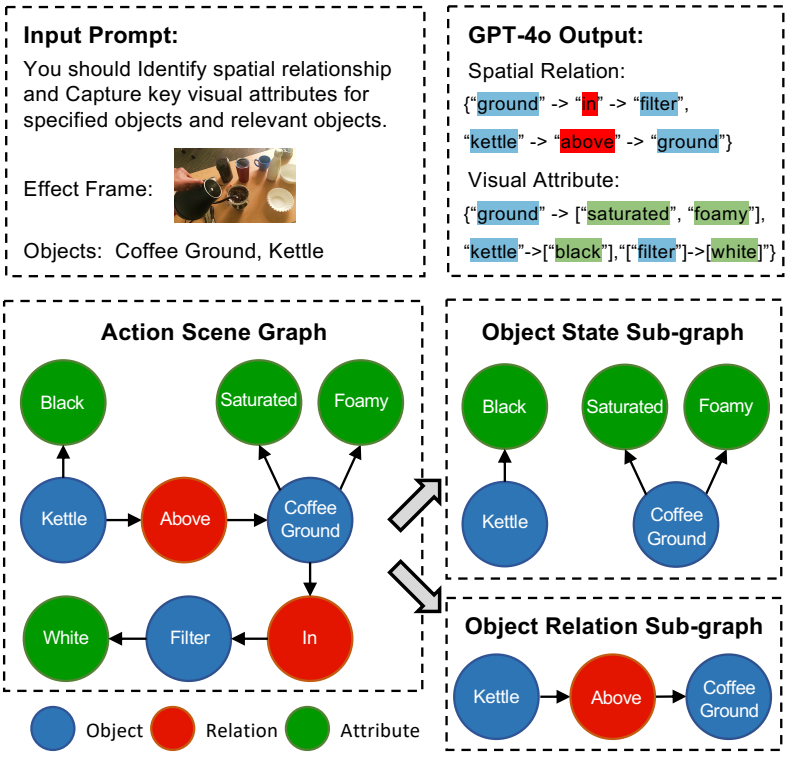

Figure 6: An example showing the construction and decomposition of action scene graph.

A.4 Solution to Failed Knowledge Extraction

During effect-frame preprocessing for multimodal feature extraction, several failure cases were observed. Grounding DINO may miss relevant objects or yield low-confidence detections, while GPT-4o may fail to recognize the image, leading to missing object states or spatial relations. To address these issues, we introduce an Effect Mask. The mask is assigned a value of 1 only when both Grounding DINO and GPT-4o successfully process the effect frame; otherwise, it is set to 0. During feature alignment, only instances with a mask value of 1 contribute to loss computation and are used to generate enhanced segment features for mistake detection. For masked-out instances, effect modeling is bypassed, and segment features from the segmentation backbone are directly fed into the mistake detector.

Algorithm 1 Action Scene Graph Construction

Input: *Relation_Dict*, *Attribute_Dict*
Output: *Nodes*, *Edges*

// Initialization.
Nodes ← dict("Object":[], "Attribute":[], "Relation":[])
Edges ← list([])

// Add object-relation edges.
for each (obj_A, rel, obj_B) in *Relation_Dict* **do**
  if $obj_A \notin$ *Nodes*$[" Object"]$ **then**
    Append obj_A to *Nodes*["Object"]
  end if
  if $obj_B \notin$ *Nodes*$[" Object"]$ **then**
    Append obj_B to *Nodes*["Object"]
  end if
  if $rel \notin$ *Nodes*$[" Relation"]$ **then**
    Append rel to *Nodes*["Relation"]
  end if
  Append (obj_A, rel), (rel, obj_B) to *Edges*
end for

// Add object-attribute edges
for each $(obj, attr)$ in *Attribute_Dict* **do**
  if $obj \notin$ *Nodes*$[" Object"]$ **then**
    Append obj to *Nodes*["Object"]
  end if
  if $attr \notin$ *Nodes*$[" Attribute"]$ **then**
    Append $attr$ to *Nodes*["Attribute"]
  end if
  Append $(obj, attr)$ to *Edges*
end for

return *Nodes*, *Edges*

Algorithm 2 Action Scene Graph Decomposition

Input: Extracted graph $Nodes$, graph $Edges$, and key objects $Objs = \{obj_i\}_{i=1}^{N}$
Output: State graph $S\text{-}Graph$, relation graph $R\text{-}Graph$

// Initialization.
Initialize $S\text{-}Graph$ ← dict({"Nodes":[], "Edges":[]})
Initialize $R\text{-}Graph$ ← dict({"Nodes":[], "Edges":[]})

// Construct the state graph from object-attribute edges
for each edge (a, b) in $Edges$ **do**
  if $a \in Objs$ **and** $b \in Nodes[" Attribute"]$ **then**
    Append a and b to $S\text{-}Graph[" Nodes"]$
    Append (a, b) to $S\text{-}Graph[" Edges"]$
  end if
end for

// Construct the state graph from object-attribute edges
for each edge (a, b) in $Edges$ **do**
  if $a \in Objs$ **and** $b \in Nodes[" Relation"]$ **then**
    Append a and b to $R\text{-}Graph[" Nodes"]$
    Append (a, b) to $R\text{-}Graph[" Edges"]$
  end if
end for

return $S\text{-}Graph, R\text{-}Graph$

For the following step in process of [goal], you should:
(1) predict relevant objects,
(2) describe resulting states of relevant objects after the action is complete. Use three concise sentences. Do not use the verb in [step].

Example A:
[goal]: Make Kimchi Fried Rice [step]: add ham
Objects: [ham, fried rice, pan]
Descriptions:
- The diced ham is mixed with fried rice.
- The ham is on the pan.
- The pan contains ham.

Example B:
[goal]: Make Pancakes [step]: pour egg
Objects: [egg, batter, bowl]
Descriptions:
- The egg is mixed with the pancake batter.
- The egg is in the mixing bowl.
- The pancake batter contains egg.

[goal]: {task} [step]: {step}

(a) Prompt used to generate relevant objects and resulting outcome descriptions.

You are an expert in spatial reasoning and visual scene understanding. Given an image, your task is to:
1. Recognize all objects in the image.
2. Identify the spatial relationship between objects in the image.
3. Capture key attributes of objects.
4. Generate a structured scene graph that accurately represents the real-world state in the image.

Important Guidelines:
1. Describe spatial relations using clear spatial terms: ("above", "below", "on", "under", "to the left of", "to the right of", "next to", "in front of", "behind").
2. Ensure each object's attributes (e.g., shape, color, material) are captured accurately.
3. Only describe what you see in the image; do not infer beyond the visual evidence.

Input Format (Image and Text):
Image: (Provided image)
Text: Generate a scene graph for the objects in the image, focusing on their relationships and attributes.

Output Format (JSON - Relations + Attributes + Sentences):

```
{
  "objects": ["<object_1>", "<object_2>", "<object_3>", ...],
  "relation": [
      {"subject": "<object_1>", "relation": "<relation>", "object": "<object_2>"},
      {"subject": "<object_1>", "relation": "<relation>", "object": "<object_3>"},
      {"subject": "<object_2>", "relation": "<relation>", "object": "<object_3>"},
      ...
  ],
  "attribute": [
      {"subject": "<object_1>", "attribute": ["<attribute_1>", "<attribute_2>"]},
      {"subject": "<object_2>", "attribute": ["<attribute_1>", "<attribute_2>"]},
      ...
  ]
}
```

(b) Prompt used to recognize object attributes and spatial relationships.

Figure 7: Prompts used in this work.

B EXPERIMENTS

B.1 IMPACT OF VLM ON PERFORMANCE

Table 4: Performance by using different VLMs for action scene generation on EgoPER dataset.

Model	Quesadilla		Oatmeal		Pinwheel		Coffee		Tea		All	
	AUC	EDA	AUC	EDA	AUC	EDA	AUC	EDA	AUC	EDA	AUC	EDA
Qwen3-VL	77.8	69.0	77.3	68.8	70.0	**62.2**	70.2	64.3	**71.4**	68.7	73.3	66.6
GPT-4o	**80.8**	68.1	77.0	68.6	69.9	61.2	70.3	66.4	71.1	69.4	73.8	66.7
GPT-5	78.9	**69.1**	**79.1**	**70.1**	**70.4**	61.9	**70.4**	**66.4**	71.3	**69.8**	**74.0**	**67.5**

At the time of completing the main paper, we employed the SOTA closed-source VLM GPT-4o (Hurst et al., 2024) and the open-source VLM Qwen3-VL (Bai et al., 2025) to generate action scene graphs. In this subsection of the Appendix, we extend Table 3f in the main paper by providing further evaluation results of our proposed method using a more recent GPT-5 model (OpenAI, 2025). Table 4 shows that the GPT-5 variant achieves the best detection performance which exceeds both models, suggesting that our method continues to benefit from ongoing advances in VLMs.

B.2 ACTION SEGMENTATION

Table 5: Impact of predicted vs. ground-truth action segmentation on EgoPER dataset.

Action Seg	Quesadilla		Oatmeal		Pinwheel		Coffee		Tea		Average	
	AUC	EDA	AUC	EDA	AUC	EDA	AUC	EDA	AUC	EDA	AUC	EDA
Pred	80.8	68.1	77.0	68.6	69.9	61.2	70.3	66.4	71.1	69.4	73.8	66.7
GT	92.8	78.4	94.2	75.0	80.7	72.0	80.8	71.3	84.8	76.7	86.7	74.7

To assess the impact of action segmentation on mistake detection, we evaluated our method by replacing predicted segments with ground-truth segments *during inference* on the test set. As shown in Table 5, using ground-truth segmentation yields a significant improvement in mistake detection performance, which can be regarded as an upper bound for the current framework. This result suggests that improvements in segmentation can consistently enhance detection performance; therefore, future work will explore leveraging more advanced action segmentation models.

B.3 PROMPT TUNING IN MISTAKE DETECTOR

Table 6: Ablation results for the learnable prompts on EgoPER dataset.

Learnable	Quesadilla		Oatmeal		Pinwheel		Coffee		Tea		All	
	AUC	EDA	AUC	EDA	AUC	EDA	AUC	EDA	AUC	EDA	AUC	EDA
✗	77.4	**68.9**	76.6	57.5	69.5	60.7	68.5	**67.0**	**72.1**	63.6	72.8	63.5
✓	**80.8**	68.1	**77.0**	**68.6**	**69.9**	**61.2**	**70.3**	66.4	71.1	**69.4**	**73.8**	**66.7**

In the mistake detector, we adopt learnable prompts combined with prefix-tuning techniques (Li & Liang, 2021) to capture action patterns. To evaluate the effectiveness of the prompt-based detector, we compare learnable prompts with fixed prompts. Both methods use the same template, "an image showing `[ACTION]` for `[TASK]`", to generate contextual embeddings. The fixed prompt does not include learnable embeddings; instead, it directly uses the frozen text encoder of EVA-02 CLIP (Fang et al., 2024) to extract features. As shown in Table 6, the learnable prompt achieves superior performance on most subtasks as well as on average compared to the fixed prompt. This indicates that fixed textual descriptions, due to the lack of temporal and contextual motion encoding, fail to align effectively with visual action features. Therefore, additional learnable embeddings are necessary to capture the temporal dynamics of actions.

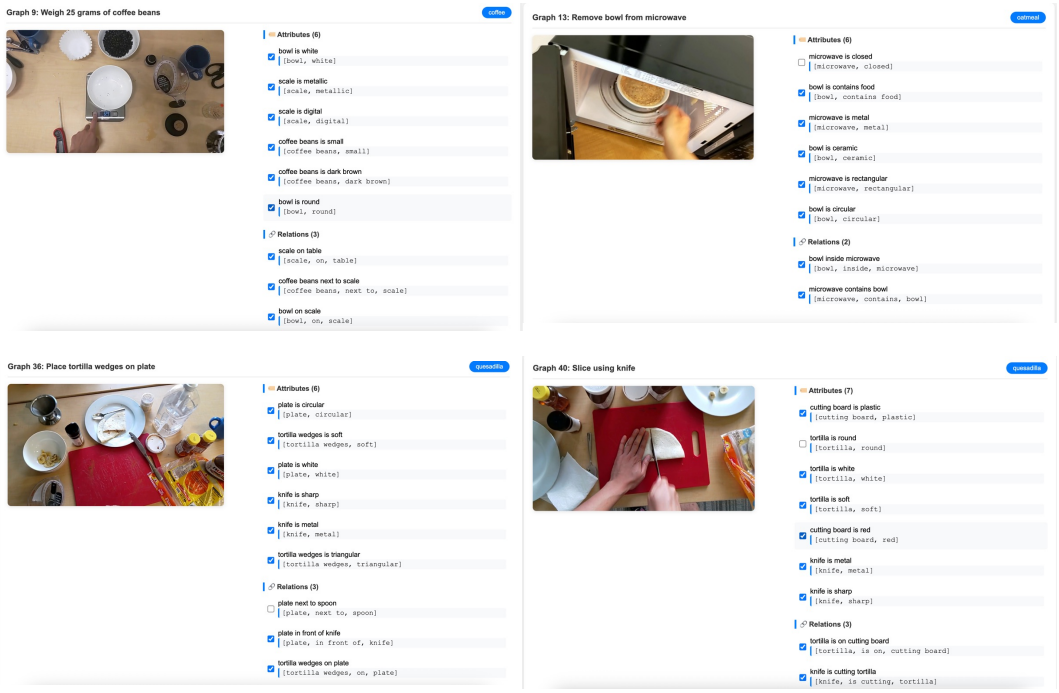

Figure 8: Web interface for scene recognition manual check.

For tasks where the performance of learnable prompts is slightly lower than that of fixed prompts, such as *Make Quesadilla*, we observed that many erroneous actions and their corresponding mistakes are less related to temporal dynamics. For example, *Place tortilla on table* (instead of on cutting board) and *Fold tortilla into a quarter-circle* (instead of a half-circle) already encode sufficient object-related information by using their textual labels. In such cases, our effect-aware model can distinguish the errors effectively, and introducing additional learnable prompts to capture action dynamics brings limited benefit and may even interfere with effect encoding, leading to performance degradation. As this work provides an initial exploration of encoding action temporal dynamics in mistake detection, extending it to a broader range of action labels is an interesting research direction.

B.4 Effect Frame Scene Recognition

In this work, we employ GPT-4o (Hurst et al., 2024) to recognize the environments of effect frames, which are then used to build scene graphs and extract symbolic knowledge. Recognition errors in this process may introduce noise into the effect modeling. Although the main paper demonstrates that effect modeling substantially improves mistake detection, we further conducted a quantitative evaluation of GPT-4o's recognition performance on complex scenes.

Specifically, we randomly sampled 20 recognition results from each sub-dataset of the EgoPER dataset (Lee et al., 2024), yielding 100 scene instances in total. As shown in Figure 8, we developed an HTML-based interface to display the recognized attributes and spatial relations for each effect frame. Two graduate students from the computer science department, with no domain conflicts, were recruited and randomly assigned 50 scenes each for manual evaluation. We define two metrics to assess recognition performance: Success Rate, the proportion of effect frames in which all recognition results are correct, and Accuracy, the proportion of correctly recognized items among all recognition results. The final scores were obtained by averaging the results of the two annotators.

Manual evaluation yielded a success rate of 48.4% and an accuracy of 87.0%. Although GPT-4o occasionally produces recognition errors, its multi-perspective scene descriptions generally complement one another, enabling reliable reconstruction of object states and spatial relationships. This provides a solid foundation for effect modeling, and we anticipate that such limitations will diminish as multimodal large language models continue to advance.

C VISUALIZATIONS

C.1 ACTION SCENE GRAPH

Figure 9 demonstrates examples of action scene graph build upon GPT-4o recognition and parsed by Algorithm 1. They captures attributes and spatial relationship of objects in a structured manner.

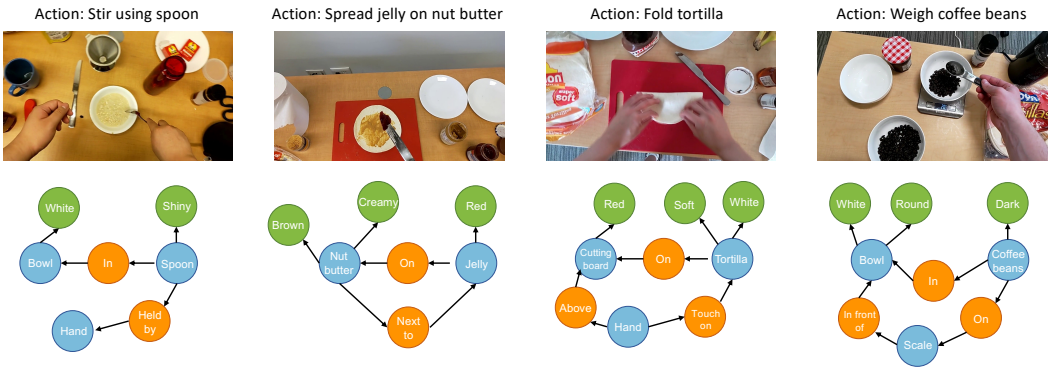

Figure 9: Examples of action scene graphs built upon effect frames.

C.2 ACTION SEGMENT FRAME SCORING

Figure 10 shows examples of the top-3 video frames with the highest weighted scores in effect-frame sampling. In our implementation, semantic and quality scores are averaged to produce the final weights. Visual inspection indicates that these frames, which are often consecutive in the video, contain highly similar content. Empirically, we find that using the top-K frames instead of only the top-1 provides negligible performance gains while substantially increasing annotation and training costs. When K is larger (e.g., $K = 5$), lower-scoring frames may introduce noisy or mismatched content, leading to reduced mistake-detection performance. Therefore, we select only the highest-scoring frame as the effect frame in our sampling strategy.

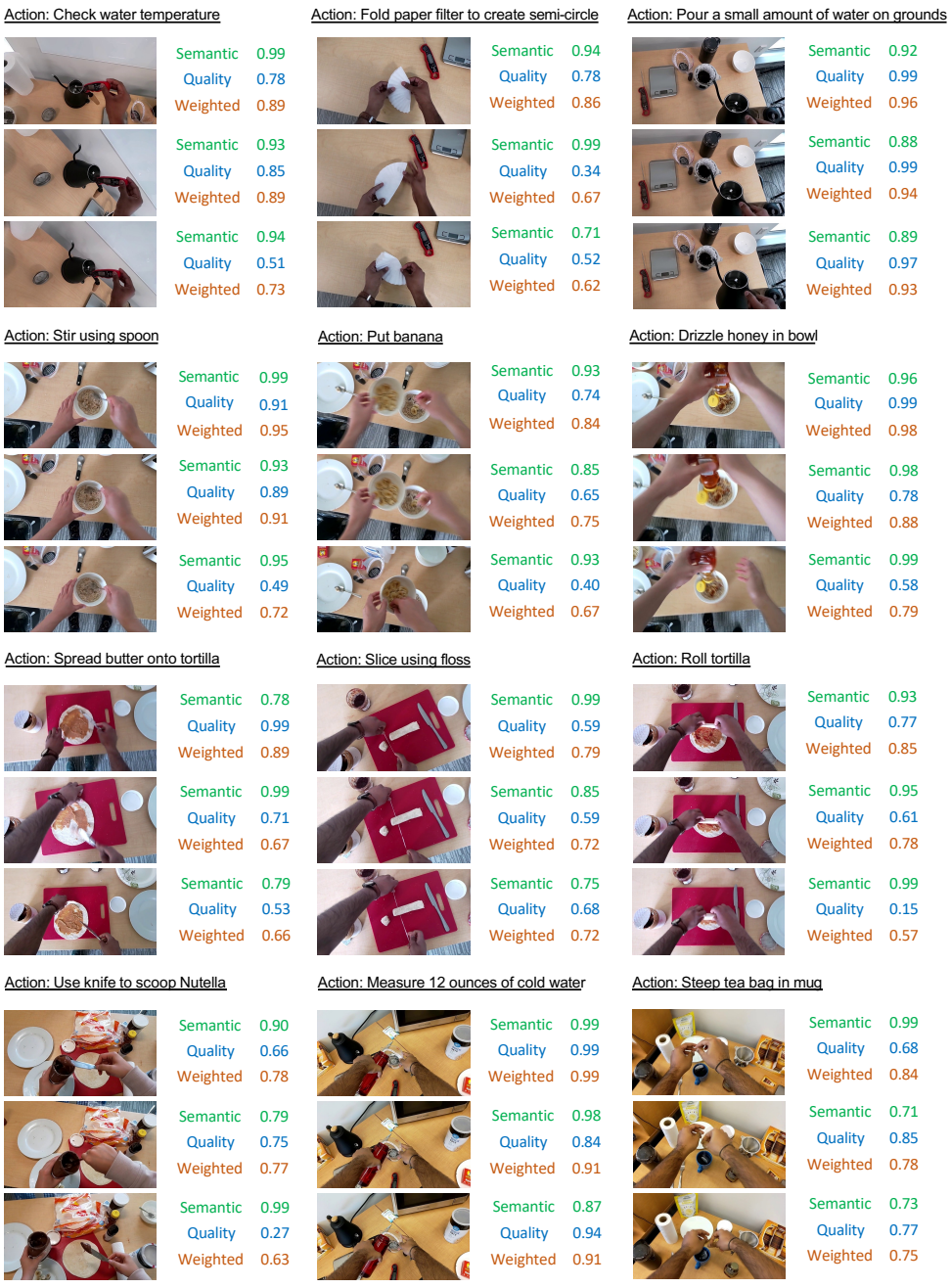

Figure 10: Examples of scores by effect frame sampling for different actions.

D FURTHER DISCUSSION

Dependence on object detection and scene graphs. The performance of our framework remains tied to the accuracy of object detection and scene graph generation. In egocentric and cluttered environments, objects are frequently occluded, blurred, or partially visible, leading to unstable detection and relationship modeling, which in turn degrades the reliability of error detection. Future directions include incorporating stronger spatiotemporal representations, such as 3D or 4D modeling, together with multimodal cues, such as depth maps or point clouds, to improve robustness under occlusion and enhance object localization and relational reasoning.

Spatial-temporal modeling. Current effect modeling focuses on single-step actions and does not explicitly capture long-range dependencies across multiple steps. In real procedural tasks, small early deviations can accumulate and lead to significant downstream errors. Future directions include modeling cross-step dependencies through hierarchical task graphs, causal reasoning structures, or program-level state transition models, thereby extending the framework to long-horizon, multi-step reasoning and enabling more comprehensive detection of complex procedural errors.

Supervision of effect modeling. Our framework currently relies on multimodal supervision generated by large pretrained models, which may hinder generalization to new domains or resource-constrained settings. To reduce this dependency, future work may explore weakly supervised or self-supervised signals, for example, leveraging temporal consistency, action outcome comparisons, or environment state transitions to automatically construct supervision, thereby improving scalability in broader domains and scenarios.

Dependence on large models. Our method relies on large models to generate pseudo-labels or auxiliary signals for action effect modeling, yet their outputs can be stochastic and inconsistent. For instance, in experiments, we observed that an identical effect frame may yield slightly different spatial relationship descriptions across runs, and such noise can propagate into learned effect representations and impact training stability. Potential mitigation strategies include adopting open-source models with reproducible checkpoints and fixed random seeds, as well as introducing consistency constraints, cross-model voting, or automatic filtering mechanisms to reduce noise propagation.

Interpretability of mistakes. Most existing methods, including ours, formulate error detection as binary classification, which provides limited insight into the underlying causes of errors. Enhancing interpretability therefore represents an important research direction. One possible solution is to integrate LLMs or LVLMs to generate human-understandable explanations, improving the transparency and usability of error detection systems.

Potential of world models. World models have recently emerged as an active research direction, as they capture environment dynamics and predict future scenes based on historical observations. These capabilities make them promising for procedural video mistake detection, particularly for online and one-class classification (OCC) settings. Future directions include predicting future states in pixel or latent space conditioned on past observations and comparing them with actual outcomes to identify inconsistencies, which correspond to procedural errors.

E USE OF LARGE LANGUAGE MODEL

In this work, large language models (LLMs) were employed solely for English translation and language polishing, with the goal of improving fluency and clarity of the manuscript so that the contributions can be presented to readers more effectively. LLMs were **not** used for idea generation, conceptualization, or any other aspect of the research.