# TGPED: TASK-GENERALIZED PROCEDURAL ERROR DETECTION AND EXPLANATION USING LARGE LANGUAGE MODELS WITH PROCEDURAL GROUNDING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

In this paper, we address temporal action segmentation, error detection, and error explanation within a single unified framework across multiple procedural tasks. Existing approaches typically require a separately trained model to segment actions for each task, limiting their scalability and practicality. Moreover, due to the limited availability of training videos containing errors, prior methods are often trained only on normal videos, preventing direct optimization for error detection and explanation. To address these limitations, we propose TGPED, a task-generalized framework for procedural error detection and explanation that avoids task-specific retraining while enabling direct optimization for error understanding. First, TGPED generates dense video captions and introduces Procedure-Grounded Temporal Segmentation (PGTS), which performs segmentation using Large Language Models (LLMs) together with generated procedural references that encode task-specific information, such as action semantics and valid action orderings. Second, we design a data generation pipeline to synthesize realistic sequences of erroneous actions, forming a cross-task training dataset. Finally, we formulate the generated sequences as inputs for fine-tuning an LLM-based error detector to jointly perform error detection and explanation. We evaluate TGPED on EgoPER and CaptainCook4D, achieving state-of-the-art performance on EgoPER and strong results on CaptainCook4D. Code will be released upon acceptance.

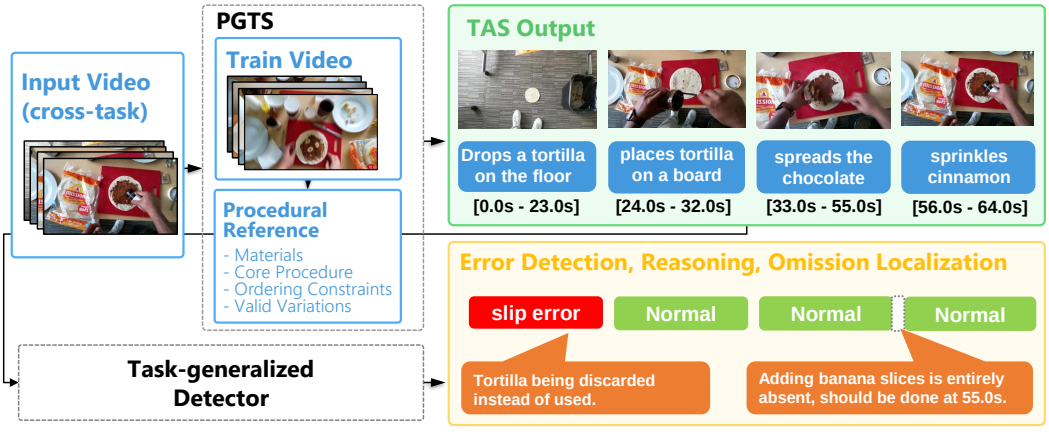


Figure 1: The illustration of our TGPED. TGPED uses a shared procedural reference for training-free action segmentation with PGTS and for error detection, explanation, and omission localization with a shared cross-task detector.

## 1 INTRODUCTION

Learning how to perform procedural activities, which consist of multiple steps and can be performed with substantial variations, often requires considerable time and effort. To facilitate this learning



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

process, people increasingly rely on AI task assistants to provide guidance and relevant information. Imagine that you are trying to make Cacio e Pepe. You search for a recipe online and find that there are several different ways to prepare it. You then decide to try multiple versions to determine which one best suits your preferences. In such a scenario, an AI task assistant needs to understand diverse procedural activities and provide useful guidance simultaneously, such as localizing and recognizing actions through temporal action segmentation (TAS) ((Gökay et al., 2025; Shen & Elhamifar, 2025; Lu & Elhamifar, 2025; 2024; Shen & Elhamifar, 2024; Farha & Gall, 2019a; Li et al., 2020; Yi et al., 2021; Souri et al., 2021; Rohrbach et al., 2012; Lea et al., 2017; Yeung et al., 2018; Zhang et al., 2022; Li et al., 2022; Liu et al., 2023)), predicting future actions ((Lin et al., 2022; Zhong et al., 2023; Chen et al., 2024)), and detecting and explaining errors when they occur ((Wang et al., 2023; Ghoddoosian et al., 2023; Ding et al., 2023; Flaborea et al., 2024b; Lee et al., 2024a; Luigi Seminara, 2024; Huang et al., 2025a; Patsch et al., 2025a; Lee et al., 2026)). Among these capabilities, TAS and error detection and explanation (EDE) are fundamental for understanding what users are doing and whether they are doing it correctly.

The standard pipeline adopted by existing EDE approaches ((Lee et al., 2024b; Lee & Elhamifar, 2025; Huang et al., 2025b; Lee & Elhamifar, 2026)) first trains TAS models to segment videos into individual actions. This step is crucial because it provides 1) the temporal boundaries on which subsequent analysis should focus and 2) action labels that serve as anchor points for determining whether errors occur. Once the action segments are obtained, existing methods apply different techniques, such as prototype learning ((Lee et al., 2024b)), video grounding with task graphs ((Lee & Elhamifar, 2025)), or reasoning with Multimodal Large Language Models (MLLMs) ((Lee & Elhamifar, 2026)). Specifically, EgoPED ((Lee et al., 2024b)) learns action prototypes and detects errors by measuring the distances between action segments and their corresponding prototypes using predefined thresholds. GTG2Vid ((Lee & Elhamifar, 2025)) constructs generalized task graphs to ground videos containing errors and identifies unmatched action segments as potential errors. AXG-Reasoner ((Lee & Elhamifar, 2026)) further decomposes actions into subactions to identify more informative keyframes and provide richer cues in prompts for MLLMs, thereby improving EDE performance. Although existing approaches achieve promising performance in both TAS and EDE, they still suffer from two major limitations.

First, existing approaches require a separate TAS model for each procedural task, meaning that a newly trained TAS model is needed whenever a new task is introduced. Moreover, collecting sufficient training videos with detailed action annotations to train a reliable TAS model is costly and time-consuming. These requirements substantially limit the scalability and practicality of existing EDE methods. Second, previous methods typically follow a one-class setting, using only normal (error-free) videos during training. Although promising EDE performance has been demonstrated under this setting, the absence of error examples during training prevents direct end-to-end optimization for EDE. Incorporating videos containing errors could enable models to directly learn to detect and explain different types of errors, while also capturing the relationships between actions and errors when they are interleaved within the same video.

To overcome these limitations, we aim to develop a single generalized framework that jointly addresses TAS and EDE across diverse procedural tasks while directly learning from both normal and erroneous executions. We therefore propose **TGPED**, a task-generalized framework for procedural error detection and explanation across multiple procedural tasks. TGPED first applies a dense video captioning model to obtain temporally localized descriptions of the input video. We then introduce **Procedure-Grounded Temporal Segmentation (PGTS)**, a training-free TAS module that constructs a task-specific procedural reference from normal demonstrations and uses a frozen LLM to align the localized descriptions with the actions defined by the reference. The resulting action segments are subsequently passed to a LLM as a shared error detector, which reasons over the observed execution with respect to the corresponding procedural reference to identify errors and generate explanations. To enable optimization for detecting and explaining errors, we develop a generation pipeline to construct training sequences consisting of normal action and error textual descriptions and explanation using normal training videos by synthesizing multiple error types for each action. Finally, we use it as a compact cross-task training set for supervised fine-tuning of the shared detector. To comprehensively show the effectiveness of our proposed framework and baselines on EDE, we introduce new metrics to evaluate error explanation and omission error detection.

Overall, our main contributions are summarized as follows:

- We propose **TGPED**, a generalized framework for TAS and EDE that handles multiple procedural tasks within a shared pipeline, without using a separate model for each task.
- To perform TAS on given videos, we introduce **Procedure-Grounded Temporal Segmentation (PGTS)**, a training-free TAS method that constructs a procedural reference from videos by using MLLMs to generate video captions with timestamps. We then use LLMs to produce task-specific procedural reference which is included into the prompt to perform TAS using LLMs.
- To enable direct optimization for EDE under one-class setting, we develop a data generation pipeline that synthesizes multiple error types for each action together to produce cross-task training sequences interleaved with normal and erroneous textual descriptions. We then train an LLM to perform EDE based on the produced dataset.

## 2 Related work

**Procedural Error Detection.** Multiple existing error detection methods utilize TAS models to segment actions, followed by an error detection approach to detect if the actions contain errors. In particular, EgoPED Lee et al. (2024b) learns normal action prototypes for detecting erroneous actions. PREGO Flaborea et al. (2024a) performs online action detection first, followed by LLMs to detect errors. DTGL Seminara et al. (2024) also detects actions and utilized learned task graph to capture errors. GTG2Vid Lee & Elhamifar (2025) aligns a generalized task graph with videos to drop the actions as detected errors. More recently, AXG-Reasoner Lee & Elhamifar (2026) decomposes actions into subactions to better detect and explain the errors within the actions using VLMs. Despite these advances, they all require task-specific TAS training for each task, limiting practical deployment across heterogeneous tasks.

**Procedural video understanding using VLM & LLM.** A growing line of work decouples visual perception from higher-level language reasoning by translating visual observations into textual representations. Socratic Models (Zeng et al., 2023) composes pretrained vision-language models (VLMs) and large language models (LLMs) using language as a shared interface. For long videos, LLoVi Zhang et al. (2024) converts short clips into textual descriptions using a VLM and aggregates them with an LLM for long-range reasoning, while LangRepo Kahatapitiya et al. (2025) maintains long video histories as compact textual representations for subsequent LLM reasoning. This paradigm has also been applied to multi-step activities. PALM Kim et al. (2024) combines VLM-derived visual context with action history for LLM-based long-term action anticipation, and similar VLM–LLM pipelines have been studied for multi-step procedural assistance (Verghese et al., 2025). In error detection, MistSense Patsch et al. (2025b) uses an LLM to explain detected mistakes, while AXG-Reasoner Lee & Elhamifar (2026) directly uses a VLM to detect and explain errors from localized observations. Existing error detection methods therefore remain focused on local evidence rather than reasoning over the complete procedural execution.

## 3 Method

### 3.1 Problem Definition

Let $\mathcal{T}$ denote a collection of procedural tasks. For a given task $\tau \in \mathcal{T}$, we consider a procedural video $V$ depicting an execution of $\tau$. We define the procedural sequence for $V$ as a sequence of $N$ normal (error-free) action segments,

$$\mathcal{A}(V) = \{(a_i, s_i, e_i)\}_{i=1}^{N}, \tag{1}$$

where $a_i$ denotes the action class of the $i$-th segment, and $s_i$ and $e_i$ denote its start and end times, respectively. Our goal is to develop a unified framework that, across multiple procedural tasks, (1) localizes both normal and erroneous actions by identifying their start and end times, and (2) explains the detected errors. We additionally consider omission errors, for which no corresponding action segment is observed in the video. To localize such errors, the model predicts the omitted action class together with the time at which the action should have occurred.

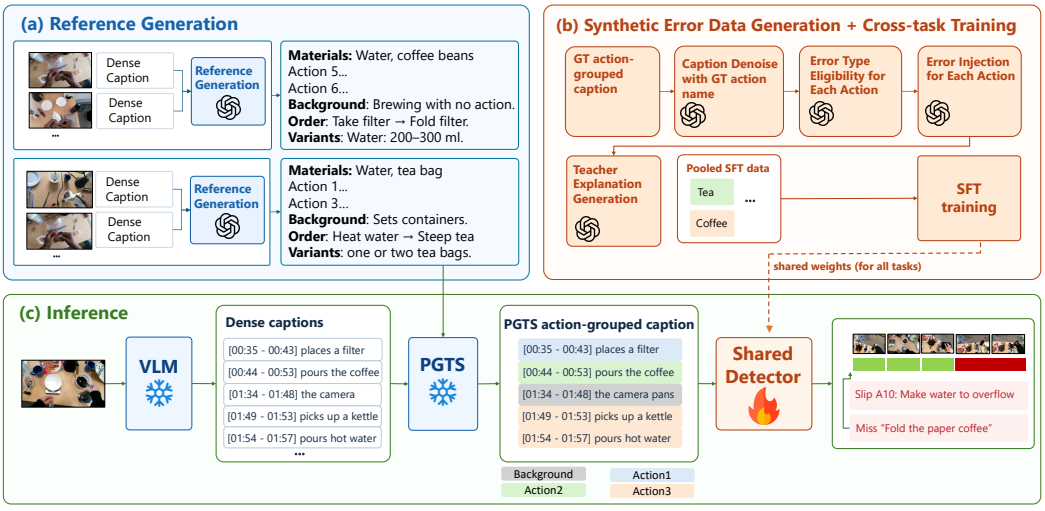


Figure 2: The framework of our TGPED. It consists of (a) Procedural Reference Generation for traning-free TAS, (b) cross-task dataset generation pipeline to synthesize training sequences and our LLM-based detector training, and (c) the inference pipeline.

### 3.2 Framework Overview

Figure 2 shows our pipeline. An MLLM first converts every video into dense video captions (DVC). (a) From the DVC of normal training videos of task $\tau$, PGTS builds a task-specific procedural reference, which it then uses at test time to prompt an LLM to assign an action class to each caption, yielding action-grouped captions without any TAS training. (b) From the ground-truth action-grouped captions of normal training videos, we synthesize a cross-task dataset of normal and erroneous caption sequences, with explanation targets written by a stronger teacher LLM. (c) We fine-tune a single LLM-based detector on this dataset, which reasons over the action-grouped captions against the procedural reference to perform EDE.

### 3.3 Procedure-Grounded Temporal Segmentation

Our goal is to develop a framework to perform TAS on a given video from any task without using a separately trained TAS model. Therefore, we introduce PGTS, a training-free method that perform TAS using DVC and a generated procedural reference for each task.

**Dense Video Captioning.** We use an MLLM to generate dense video captions for all videos. Each video is split into non-overlapping 90-second windows sampled at 1 fps, and each window is captioned in a single prompt. The prompt interleaves text and images: the frames appear in temporal order, and each frame image is preceded by a text line giving its index in the window and its timestamp in the video, so the input to a window is the sequence "`Frame 1 (timestamp 01:30) <image> Frame 2 (timestamp 01:31) <image> ...`", followed by the instruction. The instruction reads *"Your task is to produce a detailed timeline of this clip. [...] Cut a new segment whenever the action, the object being handled, or the tool changes."*, and the MLLM outputs consecutive segments each written as `[MM:SS - MM:SS] description`. The model reads the start and end time of each segment off the frame labels and decides where to cut and what to write. We denote the resulting caption sequence as

$$\mathcal{C}(V) = \{(c_i, s_i^c, e_i^c)\}_{i=1}^{M}, \tag{2}$$

where $c_i$ is the $i$-th caption, $[s_i^c, e_i^c]$ is its temporal interval, and $M$ is the number of caption segments produced by the MLLM for $V$, which varies across videos.

**Procedural Reference Construction.** A procedural reference specifies the ingredients, the sub-actions of each step, the ordering constraints, and allowed observational variants (left of Fig. 3). For each task, we take the generated DVC $\mathcal{C}(V)$ of $K$ training videos and their ground-truth action

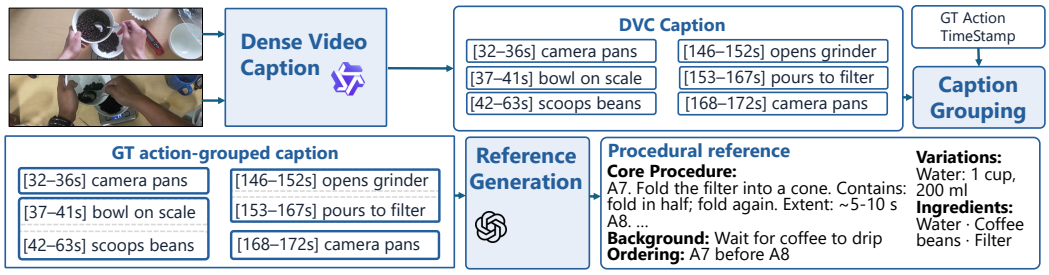


Figure 3: Our procedural reference generation pipeline. DVC captions are first grouped using GT action segments. Given multiple action-grouped captions from training videos, an LLM summarizes the task into a procedural reference containing the core procedure, background actions, ordering constraints, ingredients, and variations.

segments $\mathcal{A}(V)$. A function $F$ assigns each dense caption to the segment with the largest temporal overlap if the segments cover over half of the caption's interval; otherwise, the caption is treated as background. Captions assigned to the same segment are aggregated into ground-truth action-grouped captions $\mathcal{G}^{\mathrm{GT}}(V) = F(\mathcal{A}(V), \mathcal{C}(V))$. Finally, an LLM generates the procedural reference $R_\tau$ for task $\tau$ from $\mathcal{G}^{\mathrm{GT}}(V)$ of the $K$ videos with the prompt: *"Write one reference with exactly one step per action id, the sub-actions each step contains, ordering constraints that list only direct dependencies, and allowed observational variants that a correct execution could also show."*

**Temporal Action Segmentation.** At test time, given a test video from task $\tau$, we generate its DVC $\mathcal{C}(V)$. We then input $\mathcal{C}(V)$ and procedural reference $R_\tau$ into a LLM to obtain action-grouped captions $\mathcal{G}^{\mathrm{PGTS}}(V) = \{(\hat{a}_j, \hat{s}_j, \hat{e}_j, \mathcal{C}_j)\}_{j=1}^{\hat{N}} = \mathrm{LLM}(\mathcal{C}(V), R_\tau)$ with the prompt: *"Using the procedural reference, assign an action label to each caption in the sequence."*, where $\mathcal{C}_j$ denotes the captions grouped into the $j$-th predicted action segment $(\hat{a}_j, \hat{s}_j, \hat{e}_j)$. Note that $\mathrm{PGTS}(V)$ and $\mathcal{G}^{\mathrm{GT}}(V)$ share the same output structure. Each predicted action segment consists of start and end times, action class, and the aggregated captions describing frames within the interval.

### 3.4 Learning for Error Detection and Explanation

In this section, we describe (1) the generation pipeline for the cross-task dataset with sequences of normal and erroneous actions, and (2) the SFT method for the LLM-based error detector.

**Cross-task Dataset Generation with Pseudo Errors.** Note that we only use normal videos during training as in prior work. To optimize error detection and explanation, we therefore construct a generation pipeline to generate sequences of action and pseudo error descriptions. To begin, we utilize ground-truth action-grouped captions $\mathcal{G}^{\mathrm{GT}}(V)$ generated in Section 3.3 and use an LLM, given the ground-truth action name, refine noisy captions within each action group before synthesizing pseudo errors, with: *"Repair the lines that states something the ground-truth action rules out."*

Next, we synthesize errors that are reasonable and realistic in the procedural sequence. For each action segment in $\mathcal{G}^{\mathrm{GT}}(V)$, which represents the procedural sequence of a video, we use an LLM to determine what types of errors can be meaningfully applied, given only its refined captions and the error type definitions, under the rule: *"A type is eligible only if at least one caption line shows the detail that type would change [. . . ], so that changing a few of its words would state the error as a visible, observed fact."* Note that the error type definition is given for each dataset. We then select $p\%$ of actions to be replaced by errors in each procedural sequence, with $p \in \{25, 50, 75\}$, where the error types are drawn in a balanced round-robin fashion and the action for each type is selected uniformly among the applicable ones (see our appendix for details). Omissions and order errors are applied by removing or swapping action segments, for each other selected action, we change its refined captions using an LLM and the procedural reference $R_\tau$ with the prompt: *"Rewrite the captions so that the video contains the following errors, each in the step and lines given. Keep every other line exactly as written, except for minimal edits to other captions that name an item changed by the error"*, yielding action-grouped captions with errors $\widetilde{\mathcal{G}}^{\mathrm{GT}}(V)$.

**SFT for Error Detection and Explanation.** Our final step is to optimize an LLM-based detector, $f_\theta$, for detecting and explaining errors. For each training sample of task $\tau$, the SFT input consists of the procedural reference $R_\tau$ and the action-grouped captions with errors $\widetilde{\mathcal{G}}^{\mathrm{GT}}(V)$. The synthesis record gives the verdict and error types of every action and each omitted action, but no natural-language explanation. We therefore use a stronger LLM as a teacher to write the target answer. The teacher receives the same prompt as the detector, *"Judge, block by block, whether the person carries out the procedure correctly, then whether any step of the reference was left out."*, together with the synthesis record, and explains each error and omission in its own words, grounded in the captions and the procedural reference. An answer is retained only if it reproduces the synthesis record; otherwise it is sampled again. The retained answers serve as SFT targets for $f_\theta$

- **Action blocks:** $\hat{N}$ output blocks `<step`$j$`>`, one for each step block $g_j$ in $\mathcal{G}(V)$, in the same order and including background blocks. A correct block contains only `Verdict: normal`; an erroneous block contains `Verdict: error`, an `ErrorType` from the dataset taxonomy (on CaptainCook4D, possibly several), and a one-sentence `Explanation` of what the person did and what the reference requires.
- **Omission block:** a final block `<final>` listing all predicted omitted actions by action-class ID (`OmittedActions`), the time at which each should have begun (`InsertAt`, in mm:ss), and a one-sentence `Explanation` for each that also states the step block it belongs after; all three are `none` when no action is omitted.

Training examples from multiple tasks are jointly used to fine-tune the shared detector.

$$f_\theta(\mathcal{G}(V), R_\tau) = \Big( \{(\hat{y}_j, \hat{z}_j)\}_{j=1}^{\hat{N}}, \quad \big\{(\hat{a}_m^{\mathrm{omit}}, \hat{t}_m, \hat{z}_m^{\mathrm{omit}})\big\}_{m=1}^{\hat{M}} \Big), \tag{3}$$

where $\hat{y}_j$ denotes the error prediction (verdict and error type) for the $j$-th step block, $\hat{z}_j$ denotes its generated natural-language explanation when an error is detected, and $\hat{a}_m^{\mathrm{omit}}$, $\hat{t}_m$ and $\hat{z}_m^{\mathrm{omit}}$ denote the action class, predicted timestamp and explanation of the $m$-th omission, respectively.

# 4 Experiments

## 4.1 Experimental Setup

### 4.1.1 Dataset

We evaluate our method on EgoPER Lee et al. (2024b) and CaptainCook4D Peddi et al. (2024), following the task selection and data splits used by prior work Lee & Elhamifar (2026). EgoPER contains 386 egocentric videos across 5 cooking tasks: Coffee, Oatmeal, Pinwheels, Quesadilla, and Tea, with 5 types of errors. CaptainCook4D contains 384 egocentric recordings spanning 24 cooking tasks and includes 7 types of errors. Following prior work Lee & Elhamifar (2026), we evaluate on five tasks: Spiced Hot Chocolate, Microwave Egg Sandwich, Breakfast Burritos, Ramen, and Cucumber Raita. For omission localization, we additionally use the ground-truth task graphs to derive the temporal locations of omitted actions.

### 4.1.2 Evaluation Metrics

We evaluate our method along four dimensions: temporal action segmentation, error detection, error explanation, and omission localization. For TAS, we report F1@0.5, Edit score, and Accuracy following Farha & Gall (2019b). For error detection, we report segment-level F1@0.5, AUC, and EDA as in Lee et al. (2024a); Lee & Elhamifar (2026). With ground-truth action segments, we classify each segment as normal or erroneous following Lee & Elhamifar (2026) and report F1 as the mean of the normal- and error-class F1, each averaged over videos. For error explanation, we use the similarity score of Lee & Elhamifar (2026) and additionally introduce a pairwise preference evaluation. We run both models on the ground-truth action segments and use GPT-5.6-terra as the judge. For each ground-truth error segment that both models flag as erroneous, the judge compares their explanations against a question constructed from the annotated step and error description and selects the better one. Each pair is evaluated twice with the order reversed to reduce position bias, and we report the win rate over these segments. We also introduce a new metric for omission error detection, since the previous metric Lee et al. (2024a) ignores the temporal locations of omissions. We

match predicted omissions to ground-truth omitted actions by action class, and a matched prediction is a true positive at threshold $\theta$ only if $|\hat{t} - t^\star| \leq \theta L_v$, where $\hat{t}$ is its predicted time and $L_v$ is the video length. Other predictions and ground-truth omissions without a true positive count as false positives and false negatives, respectively. We compute localization F1 per task and report its mean over tasks at $\theta \in \{1\%, 2.5\%, 5\%, 10\%\}$. Additional details for all evaluation metrics are provided in the supplementary material.

#### 4.1.3 BASELINES

We mainly compare with baselines that perform both TAS and error detection, following AXG-Reasoner Lee & Elhamifar (2026). For TAS, we compare with EgoPED Lee et al. (2024b), AMNAR Huang et al. (2025b) and GTG2Vid Lee & Elhamifar (2025). For error detection, we compare with EgoPED, AMNAR Huang et al. (2025b), State-Change CF Kung et al. (2025), GTG2Vid, AEM Guo et al. (2026), and AXG-Reasoner. For error explanation, we compare with AXG-Reasoner, which we reproduce with Qwen3.6-27B, which is the VLM used in our pipeline, instead of its original Qwen2.5-VL-32B. For omission localization, since no prior method localizes omissions, we derive them from the segmentation of EgoPED, AMNAR, and GTG2Vid with the ground-truth task graph. For GTG2Vid, each node dropped by its alignment is an omission at the frame where it is dropped. For the others, task-graph actions absent from the predicted segments are omissions, placed at the end of the latest-ending performed predecessor, or at time 0 if none. We also report a zero-shot variant, TGPED (ZS), which keeps PGTS but replaces the trained detector with a frozen LLM.

#### 4.1.4 IMPLEMENTATION DETAILS

We use Qwen3.6-27B for dense video captioning and PGTS (non-overlapping 90-s windows, 1 FPS) and GPT-5.6 Terra for procedural reference construction from all normal training videos. For each normal video, we synthesize two pseudo-error variants per error density $\rho \in \{0.25, 0.5, 0.75\}$, each with $\mathrm{round}(\rho N)$ errors spread evenly over error types (including omission) on uniformly drawn eligible actions. GPT-5.6 Terra also selects eligible error types, writes the modifications, and, as the teacher, writes explanation targets from the synthesis record. We fine-tune Qwen3.5-9B with LoRA ($r$=16, $\alpha$=32), learning rate $2 \times 10^{-4}$, and effective batch size 16 for 3/2 epochs on 1,302/224 examples (EgoPER/CaptainCook4D). Details are in the appendix.

### 4.2 EXPERIMENTAL RESULTS

#### 4.2.1 TEMPORAL ACTION SEGMENTATION RESULTS

Table 1 reports TAS results. Without any training, PGTS achieves 70.1% Acc on EgoPER, surpassing AMNAR and GTG2Vid. On CaptainCook4D, where training data are scarce, PGTS outperforms GTG2Vid on all three aggregate metrics, improving Acc, Edit, and F1 by 27.9, 14.5, and 19.1 points. PGTS lags supervised methods in Edit and F1 on EgoPER, as 45.4% of its background segments are shorter than 2 s, and the average length in DVC captioning is 6s. Such short segments are penalized by Edit and F1 but account for few frames, so Acc is barely affected.

#### 4.2.2 ERROR DETECTION RESULTS

Table 2 reports error detection with predicted segments. On EgoPER, TGPED achieves the best F1 (42.5). As its zero-shot variant matches AXG-Reasoner (30.1 vs. 30.3), the gain comes from fine-tuning on pseudo-errors, which raises normal/error-class F1 from 38.4/21.9 to 49.2/35.9. On CaptainCook4D, TGPED, its zero-shot variant, and AXG-Reasoner are comparable (31.0, 31.5, 30.2), well above other baselines.

Table 3 reports EDA and AUC. On EgoPER, TGPED attains the highest EDA (79.5, +6.8 over State-Change) and higher AUC (65.1) than EgoPED and State-Change. Fine-tuning improves EDA/AUC by 11.2/10.2 on EgoPER and 2.9/2.3 on CaptainCook4D.

Table 4 uses ground-truth segments, where TGPED achieves the best F1 on both datasets (68.4 vs. 62.1 for AXG-Reasoner on EgoPER; 50.3 vs. 41.9 on CaptainCook4D). SFT gains 12.0 on EgoPER, consistent with the predicted-segment setting (12.4), and 4.2 on CaptainCook4D, where segmentation

Table 1: Temporal action segmentation results on EgoPER and CaptainCook4D.

| Method | Training-Free | EgoPER | | | CaptainCook4D | | |
|---|---|---|---|---|---|---|---|
| | | Acc | Edit | F1 | Acc | Edit | F1 |
| EgoPED | No | **75.9** | **69.7** | 48.9 | – | – | – |
| AMNAR | No | 68.9 | 63.1 | 50.4 | – | – | – |
| GTG2Vid | No | 66.8 | 61.1 | **60.0** | 47.2 | 60.2 | 37.2 |
| **PGTS** | **Yes** | 70.1 | 57.6 | 45.0 | **75.1** | **74.7** | **56.3** |

Table 2: Segment-level error detection results on EgoPER and CaptainCook4D.

| Method | TAS | EgoPER | | | CaptainCook4D | | |
|---|---|---|---|---|---|---|---|
| | | N. | E. | F1 | N. | E. | F1 |
| EgoPED | ACTF | 24.0 | 21.1 | 22.5 | 13.4 | 7.5 | 10.5 |
| GTG2Vid | GTG2Vid | 41.7 | 22.7 | 32.2 | 17.9 | 15.6 | 16.8 |
| AXG-Reasoner | GTG2Vid | 33.1 | 27.4 | 30.3 | 20.9 | 39.6 | 30.2 |
| TGPED (ZS) | PGTS | 38.4 | 21.9 | 30.1 | **22.8** | 40.2 | **31.5** |
| **TGPED** | PGTS | **49.2** | **35.9** | **42.5** | 20.1 | **41.8** | 31.0 |

Table 3: Frame-level error detection on EgoPER and CaptainCook4D, measured by EDA and frame-wise AUC.

| Method | EgoPER | | CaptainCook4D | |
|---|---|---|---|---|
| | EDA | AUC | EDA | AUC |
| EgoPED | 57.0 | 62.0 | – | – |
| AMNAR | 64.4 | 68.5 | – | – |
| State-Change CF | 72.7 | 60.5 | – | – |
| AEM | 66.7 | **73.8** | **64.9** | **61.5** |
| TGPED (ZS) | 68.3 | 54.9 | 60.3 | 51.9 |
| **TGPED** | **79.5** | 65.1 | 63.2 | 54.2 |

Table 4: Error detection on ground-truth action segments for EgoPER and CaptainCook4D. For TGPED variants, we form GT action-grouped captions on GT segment timestamps.

| Method | EgoPER | | | CaptainCook4D | | |
|---|---|---|---|---|---|---|
| | N. | E. | F1 | N. | E. | F1 |
| AXG-Reasoner | 79.6 | 44.6 | 62.1 | 20.6 | **63.3** | 41.9 |
| TGPED (ZS) | 74.7 | 38.2 | 56.4 | 36.2 | 55.9 | 46.1 |
| **TGPED** | **89.5** | **47.3** | **68.4** | **47.7** | 52.8 | **50.3** |

errors had masked its benefit. AXG-Reasoner's high error-class F1 on CaptainCook4D comes at only 20.6 normal-class F1, whereas TGPED stays balanced.

#### 4.2.3 Error Explanation Results

Table 5 reports the explanation score of AXG-Reasoner, which averages an LLM similarity score over ground-truth error segments and assigns zero to missed ones. On EgoPER, TGPED reaches 30.6, above AXG-Reasoner with Qwen3.6-27B (24.9) and its zero-shot variant (25.2). On CaptainCook4D, TGPED (18.1) falls below AXG-Reasoner (23.2) and its zero-shot variant (20.1). Because this score penalizes missed segments but not false alarms, it mixes detection coverage with explanation quality.

Table 6 isolates explanation quality by comparing explanations only on error segments that both methods flag. TGPED is preferred over AXG-Reasoner on 79.1% of these segments on EgoPER and 69.2% on CaptainCook4D, so its lower CaptainCook4D explanation score reflects coverage rather than explanation quality. Against the zero-shot detector, the preference is 55.3% and 56.8%, a modest gain from fine-tuning.

#### 4.2.4 Omission Localization Results

Table 7 reports localization F1. On CaptainCook4D, TGPED achieves the best F1 at every threshold, roughly doubling the strongest baseline, GTG2Vid (e.g., 40.1 vs. 19.8 at 5%), and even the zero-shot detector outperforms all baselines at every threshold. On EgoPER, EgoPED performs best. TGPED is competitive with GTG2Vid, but it infers both from the procedural reference built from normal training videos without task graph during test time. SFT improves over the zero-shot detector at every threshold on both datasets, by 10.8 to 14.4 points on EgoPER and 8.7 to 14.4 points on CaptainCook4D.

#### 4.2.5 Quality of Pseudo Error

Figure 4 compares one of our pseudo errors with real errors. For a real error in a test video, we give the error writer only its step and error type and let it write the error into the same step of a normal video of the same task. Without seeing the real error, the writer generate the same error: on EgoPER, it pours the liquid into a second bowl and then back into the bowl of oats, reproducing both the slip and the correction; on CaptainCook4D, it changes "an egg" to "two eggs", matching the real measurement error. Each edit changes only a few words and keeps the captioner's phrasing, so the detector is trained on errors in the same form as those it detects at test time.

Table 5: Error explanation score (%) under the AXG-Reasoner protocol on EgoPER and CaptainCook4D.

| Method | EgoPER | CaptainCook4D |
|---|---|---|
| AXG-Reasoner | 24.9 | **23.2** |
| TGPED (ZS) | 25.2 | 20.1 |
| **TGPED** | **30.6** | 18.1 |

Table 6: Winning rate (%) of TGPED in pairwise explanation comparison against other methods on EgoPER and CaptainCook4D. A tie corresponds to 50. Higher is better.

| Compared Method | EgoPER | CaptainCook4D |
|---|---|---|
| AXG-Reasoner | 79.1 | 69.2 |
| TGPED (ZS) | 55.3 | 56.8 |

Table 7: Omission localization (F1@$\theta$).

| Method | EgoPER | | | | CaptainCook4D | | | |
|---|---|---|---|---|---|---|---|---|
| | 1% | 2.5% | 5% | 10% | 1% | 2.5% | 5% | 10% |
| EgoPED | **34.3** | **49.8** | **51.7** | **54.5** | 7.3 | 8.4 | 9.2 | 11.7 |
| AMNAR | 17.1 | 21.4 | 24.0 | 31.8 | – | – | – | – |
| GTG2Vid | 26.8 | 32.1 | 38.3 | 46.1 | 11.4 | 17.6 | 19.8 | 22.4 |
| TGPED (ZS) | 10.2 | 15.5 | 25.1 | 33.2 | 16.9 | 22.0 | 25.7 | 35.0 |
| **TGPED** | 21.5 | 29.9 | 39.0 | 44.0 | **25.6** | **36.1** | **40.1** | **47.4** |

Table 8: Ablation on training strategy on segment-level error detection (F1@0.5).

| Training Setting | EgoPER | CaptainCook4D |
|---|---|---|
| TGPED (ZS) | 30.1 | **31.5** |
| Task-Specific SFT | 41.7 | 29.2 |
| Leave-One-Task-Out SFT | 40.0 | 28.7 |
| **TGPED** | **42.5** | 31.0 |

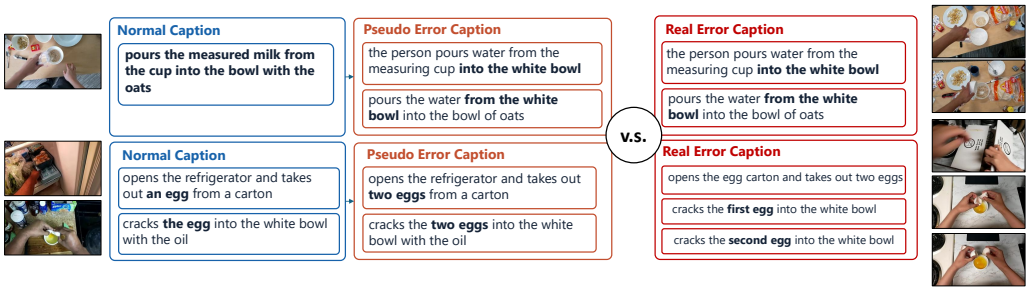


Figure 4: Pseudo errors vs. real errors of the same step and error type. Top: EgoPER Oatmeal, correction. Bottom: CaptainCook4D Microwave Egg Sandwich, measurement.

#### 4.2.6 Ablation Study

Table 8 examines whether the detector is task-general. First, a single detector trained jointly on all tasks (TGPED) performs on par with or better than five separate task-specific detectors, with an overall F1 of 42.5 vs. 41.7 on EgoPER and 31.0 vs. 29.2 on CaptainCook4D. Second, leave-one-task-out SFT, which is trained without any samples of the evaluated task, stays within 2.5 points of TGPED on both datasets and outperforms the zero-shot detector by 9.9 points on EgoPER. This indicates that the detector does not rely on having been trained on a task: given the procedural reference of an unseen task, it can still judge the captions against that reference to detect errors. In practice, this means TGPED can be deployed on new procedural tasks by simply providing their procedural references, without collecting task-specific annotations or retraining.

## 5 Conclusion and Limitations

We presented TGPED, a task-generalized framework that combines training-free Procedure-Grounded Temporal Segmentation (PGTS) with a shared LLM-based detector to perform action segmentation, error detection, explanation, and omission localization across procedural tasks without task-specific retraining. We further introduced a pseudo-error generation pipeline that enables direct optimization for error understanding using only normal training videos. Experiments on EgoPER and CaptainCook4D demonstrate strong performance across all evaluated components and show that a single detector can generalize across tasks when conditioned on the corresponding procedural reference.

**Limitations.** Since TGPED reasons over dense video captions, its performance depends on the captioner. Fine-grained details that the captioner omits or misdescribes may lead to missed errors, and this dependence is expected to lessen as MLLMs improve. In addition, PGTS operates at caption-level granularity and may merge very short background segments into adjacent steps.

### AI USE STATEMENT

In this work, we used generative AI tools to generate synthetic datasets, design and provide feedback on research methodology and experiments, clean and reformat datasets, and assist in interpreting experimental results. We did not use generative AI tools for the other required-disclosure tasks applicable to this work, and the remaining required-disclosure tasks are not applicable. We reviewed all AI-assisted outputs before incorporating them into the work. In particular, synthetic data and dataset transformations were manually inspected for consistency with the intended procedures and labels, methodological suggestions were independently evaluated by the authors before adoption, and AI-assisted interpretations of experimental results were verified against the reported quantitative results. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

### ETHICS STATEMENT

Our work generates synthetic procedural errors, which may not fully reflect the distribution and complexity of naturally occurring human errors and may introduce biases from the generative models. We do not intend the resulting models for direct deployment in safety-critical settings without further validation.

### REPRODUCIBILITY STATEMENT

We provide the complete pseudo-error data generation pipeline in the Appendix, including the prompts used for generation and the sampling configurations of the LLMs. These details are provided to facilitate reproduction of our data generation procedure and experimental results. We will release the code upon acceptance.

## REFERENCES

Joya Chen, Zhaoyang Lv, Shiwei Wu, Kevin Qinghong Lin, Chenan Song, Difei Gao, Jia-Wei Liu, Ziteng Gao, Dongxing Mao, and Mike Zheng Shou. Videollm-online: Online video large language model for streaming video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Guodong Ding, Fadime Sener, Shugao Ma, and Angela Yao. Every mistake counts in assembly. *arXiv: 2307.16453*, 2023.

Yazan Abu Farha and Jurgen Gall. Ms-tcn: Multi-stage temporal convolutional network for action segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3575–3584, 2019a.

Yazan Abu Farha and Jürgen Gall. Ms-tcn: Multi-stage temporal convolutional network for action segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3575–3584, 2019b.

Alessandro Flaborea, Guido Maria D'Amely Di Melendugno, Leonardo Plini, Luca Scofano, Edoardo De Matteis, Antonino Furnari, Giovanni Maria Farinella, and Fabio Galasso. Prego: online mistake detection in procedural egocentric videos. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18483–18492. IEEE, 2024a.

Alessandro Flaborea, Guido Melendugno, Leonardo Pliniq, Luca Scofanoq, Edoardo Matteisq, Antonino Furnari, Giovanni Farinella, and Fabio Galasso. Prego: online mistake detection in procedural egocentric videos. *IEEE Conference on Computer Vision and Pattern Recognition*, 2024b.

Reza Ghoddoosian, Isht Dwivedi, Nakul Agarwal, and Behzad Dariush. Weakly-supervised action segmentation and unseen error detection in anomalous instructional videos. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10128–10138, 2023.

Uzay Gökay, Federico Spurio, Dominik R. Bach, and Juergen Gall. Skeleton motion words for unsupervised skeleton-based temporal action segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

Wenliang Guo, Yujiang Pu, and Yu Kong. Procedural mistake detection via action effect modeling. In *International Conference on Learning Representations*, volume 2026, pp. 43009–43030, 2026.

Wei-Jin Huang, Yuan-Ming Li, Zhi-Wei Xia, Yu-Ming Tang, Kun-Yu Lin, Jian-Fang Hu, and Wei-Shi Zheng. Modeling multiple normal action representations for error detection in procedural tasks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2025a.

Wei-Jin Huang, Yuan-Ming Li, Zhi-Wei Xia, Yu-Ming Tang, Kun-Yu Lin, Jian-Fang Hu, and Wei-Shi Zheng. Modeling multiple normal action representations for error detection in procedural tasks. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 27794–27804. IEEE, 2025b.

Kumara Kahatapitiya, Kanchana Ranasinghe, Jongwoo Park, and Michael S. Ryoo. Language repository for long video understanding. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 5627–5646. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.294.

Sanghwan Kim, Daoji Huang, Yongqin Xian, Otmar Hilliges, Luc Van Gool, and Xi Wang. PALM: Predicting actions through language models. In *European Conference on Computer Vision*, pp. 140–158. Springer, 2024. doi: 10.1007/978-3-031-73007-8_9.

Chi-Hsi Kung, Frangil Ramirez, Juhyung Ha, Yi-Ting Chen, David Crandall, and Yi-Hsuan Tsai. What changed and what could have changed? state-change counterfactuals for procedure-aware video representation learning. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 12294–12306. IEEE, 2025.

C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager. Temporal convolutional networks for action segmentation and detection. *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

S. Lee, Z. Lu, Z. Zhang, M. Hoai, and E. Elhamifar. Error detection in egocentric procedural task videos. *IEEE Conference on Computer Vision and Pattern Recognition*, 2024a.

Shih-Po Lee and Ehsan Elhamifar. Error recognition in procedural videos using generalized task graph. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 10009–10021. IEEE, 2025.

Shih-Po Lee and Ehsan Elhamifar. Axg-reasoner: Error detection and explanation in long task videos with vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3421–3431, 2026.

Shih-Po Lee, Zijia Lu, Zekun Zhang, Minh Hoai, and Ehsan Elhamifar. Error detection in egocentric procedural task videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18655–18666, 2024b.

Shih-Po Lee, Reza Ghoddoosian, Faizan Siddiqui, Enna Sachdeva, and Behzad Dariush. Estanet: Efficient online error detection in procedural videos via prediction inconsistency. In *Computer Vision – ECCV 2026*, 2026.

M. Li, L. Chen, Y. Duarr, Z. Hu, J. Feng, J. Zhou, and J. Lu. Bridge-prompt: Towards ordinal action understanding in instructional videos. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, jun 2022.

Shi-Jie Li, Yazan AbuFarha, Yun Liu, Ming-Ming Cheng, and Juergen Gall. Ms-tcn++: Multi-stage temporal convolutional network for action segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020. doi: 10.1109/TPAMI.2020.3021756.

Xudong Lin, Fabio Petroni, Gedas Bertasius, Marcus Rohrbach, Shih-Fu Chang, and Lorenzo Torresani. Learning To Recognize Procedural Activities with Distant Supervision . In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Yang Liu, Jiayu Huo, Jingjing Peng, Rachel Sparks, Prokar Dasgupta, Alejandro Granados, and Sebastien Ourselin. Skit: a fast key information video transformer for online surgical phase recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 21074–21084, 2023.

Z. Lu and E. Elhamifar. Fact: Frame-action cross-attention temporal modeling for efficient action segmentation. *IEEE Conference on Computer Vision and Pattern Recognition*, 2024.

Z. Lu and E. Elhamifar. Multi-modal few-shot temporal action segmentation. *International Conference on Computer Vision*, 2025.

Antonino Furnari Luigi Seminara, Giovanni Maria Farinella. Differentiable task graph learning: Procedural activity representation and online mistake detection from egocentric videos. *Neural Information Processing Systems*, 2024.

Constantin Patsch, Yuankai Wu, Marsil Zakour, Driton Salihu, and Eckehard Steinbach. Mistsense: Versatile online detection of procedural and execution mistakes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025a.

Constantin Patsch, Yuankai Wu, Marsil Zakour, Driton Salihu, and Eckehard Steinbach. Mistsense: Versatile online detection of procedural and execution mistakes. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 14528–14537. IEEE, 2025b.

Rohith Peddi, Shivvrat Arya, Bharath Challa, Likhitha Pallapothula, Akshay Vyas, Bhavya Gouripeddi, Qifan Zhang, Jikai Wang, Vasundhara Komaragiri, Eric Ragan, et al. Captaincook4d: A dataset for understanding errors in procedural activities. *Advances in Neural Information Processing Systems*, 37:135626–135679, 2024.

M. Rohrbach, S. Amin, M. Andriluka, and B. Schiele. A database for fine grained activity detection of cooking activities. *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

Luigi Seminara, Giovanni Maria Farinella, and Antonino Furnari. Differentiable task graph learning: Procedural activity representation and online mistake detection from egocentric videos. In *Advances in Neural Information Processing Systems*, 2024.

Y. Shen and E. Elhamifar. Progress-aware online action segmentation for egocentric procedural task videos. *IEEE Conference on Computer Vision and Pattern Recognition*, 2024.

Y. Shen and E. Elhamifar. Understanding multi-task activities from single-task videos. *IEEE Conference on Computer Vision and Pattern Recognition*, 2025.

Yaser Souri, Mohsen Fayyaz, Luca Minciullo, Gianpiero Francesca, and Juergen Gall. Fast Weakly Supervised Action Segmentation Using Mutual Consistency. *PAMI*, 2021.

Mrinal Verghese, Brian Chen, Hamid Eghbalzadeh, Tushar Nagarajan, and Ruta P. Desai. User-in-the-loop evaluation of multimodal LLMs for activity assistance. In *Proceedings of the Winter Conference on Applications of Computer Vision*, pp. 1144–1154, 2025.

Xin Wang, Taein Kwon, Mahdi Rad1 Bowen Pan, Ishani Chakraborty, and Sean Andrist. Holoassist: an egocentric human interaction dataset for interactive ai assistants in the real world. *IEEE International Conference on Computer Vision*, 2023.

S. Yeung, O. Russakovsky, N. Jin, M. Andriluka, G. Mori, and L. Fei-Fei. Every moment counts: Dense detailed labeling of actions in complex videos. *International Journal of Computer Vision*, 2018.

Fangqiu Yi, Hongyu Wen, and Tingting Jiang. Asformer: Transformer for action segmentation. In *The British Machine Vision Conference (BMVC)*, 2021.

Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Marcin Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael S. Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, and Pete Florence. Socratic models: Composing zero-shot multimodal reasoning with language. In *International Conference on Learning Representations*, 2023.

Ce Zhang, Taixi Lu, Md Mohaiminul Islam, Ziyang Wang, Shoubin Yu, Mohit Bansal, and Gedas Bertasius. A simple LLM framework for long-range video question-answering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 21715–21737. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.1209.

Junbin Zhang, Pei-Hsuan Tsai, and Meng-Hsun Tsai. Semantic2graph: Graph-based multi-modal feature fusion for action segmentation in videos, 2022.

Yiwu Zhong, Licheng Yu, Yang Bai, Shangwen Li, Xueting Yan, and Yin Li. Learning procedure-aware video representation from instructional videos and their narrations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

## APPENDIX OVERVIEW

This appendix complements the main paper with the following details:

- Appendix A describes the benchmark splits of EgoPER and CaptainCook4D, frame sampling, and the dense video captioning that produces the only visual evidence used by all later stages.
- Appendix B lists the model and decoding setting of each stage, the supervised fine-tuning configuration of the detector, and the hardware.
- Appendix C details the synthesis pipeline, from caption refinement and error sampling to error writing and teacher answers, with statistics of the resulting 1,302 EgoPER and 224 CaptainCook4D training examples, a generated procedural reference, and one synthesized example per dataset.
- Appendix D defines all evaluation metrics, including TAS, error detection F1@0.5, detection on ground-truth segments, EDA and AUC, omission localization F1@$\tau$, and the pairwise explanation comparison.
- Appendix E gives the source of every baseline number and how each baseline is run or re-scored.
- Appendix F explains how the ground-truth omitted steps and their reference time $t^*$ are derived from the task graph, and how the predicted time $\hat{t}$ is obtained for the baselines and for TGPED.
- Appendix G reports the detection rate and typing accuracy per error type, together with the false alarm rate on normal segments.
- Appendix H shows TGPED's outputs on three test videos, including a failure case that illustrates four sources of error.
- Appendix I lists every prompt used by our pipeline and evaluation verbatim.

## A DATASET & PREPROCESSING DETAILS

### A.1 BENCHMARK SPLITS

EgoPER uses the official splits: 186 normal training videos and 188 test videos (173 error, 15 normal). CaptainCook4D follows AXG-Reasoner Lee & Elhamifar (2026): 32 normal training recordings and 40 error test recordings over five recipes. Only normal training videos are used to build the procedural references and the synthetic data.

### A.2 FRAME SAMPLING

Both datasets are sampled at 1 fps at $640 \times 360$. For CaptainCook4D we use the HoloLens 2 PV stream, as in AXG-Reasoner; for EgoPER we take the frame nearest to each second and resize it from $1280 \times 720$.

### A.3 DENSE VIDEO CAPTIONING

Each video is split into non-overlapping 90 s windows (a final window shorter than 3 s is merged into the previous one). Qwen3.6-27B receives the window's frames, each preceded by its timestamp ("Frame $i$ (timestamp MM:SS)"), and the captioning prompt (Appendix I.1), whose only task information is the dish name. It writes lines "[MM:SS - MM:SS] description" with greedy decoding and at most 1,536 output tokens per window. We clip every line to its window and trim overlapping lines so that each starts after the previous one ends. The concatenated timeline is the only visual evidence used by all later stages.

# B Implementation Details

## B.1 Models

Tab. 9 lists the model of each stage. The Qwen models are served with vLLM; GPT-5.6 Terra uses its default reasoning effort.

Table 9: Model and decoding per stage.

| Stage | Model | Decoding |
|---|---|---|
| Dense video captioning | Qwen3.6-27B | greedy, no thinking |
| Caption refinement (train) | GPT-5.6 Terra | default |
| Procedural reference | GPT-5.6 Terra | default |
| PGTS | Qwen3.6-27B (text only) | greedy, thinking |
| Error-type eligibility | GPT-5.6 Terra | default |
| Error writer | GPT-5.6 Terra | default |
| Teacher answers | GPT-5.6 Terra | default |
| Detector, TGPED | Qwen3.5-9B + LoRA | sampled, no thinking |
| Detector, TGPED (ZS) | Qwen3.5-9B | sampled, no thinking |

## B.2 Decoding

PGTS uses the captioning checkpoint of Qwen3.6-27B, text only, with thinking and greedy decoding. Both detectors run without thinking, with the sampling setting recommended for Qwen3.5 in non-thinking mode (temperature 0.7, top-$p$ 0.8, top-$k$ 20, min-$p$ 0, presence penalty 1.5) and at most 16,384 output tokens.

## B.3 Supervised Fine-Tuning

One detector per dataset is trained with ms-swift (Tab. 10); LoRA covers all linear layers of Qwen3.5-9B (43.3M parameters, 0.46%), with the loss on the target answer only.

Table 10: SFT settings.

| | EgoPER | CaptainCook4D |
|---|---|---|
| Training examples | 1,302 | 224 |
| Epochs | 3 | 2 |
| Learning rate | $2 \times 10^{-4}$ | |
| LR schedule | cosine, 3% warm-up | |
| Effective batch size | 16 | |
| LoRA rank / $\alpha$ / dropout | 16 / 32 / 0.05 | |
| LoRA target | all linear layers | |
| Precision | bfloat16, gradient checkpointing | |
| Seed | 42 | |

## B.4 Hardware

All models run on nodes with eight NVIDIA H100 or H200 GPUs. SFT runs data-parallel on four H100 GPUs.

# C Synthetic Data Details

## C.1 Synthesis Pipeline

Synthetic examples are built from normal training videos, whose captions are grouped by the ground-truth segments (a line joins the action covering most of it, otherwise background).

**Caption refinement.** So that misread details are not learned as normal, GPT-5.6 Terra is given each group's captions and its annotated action, and replaces only the words the action rules out: a wrong number, amount, or setting, or a wrong or vague item ("red powder" for paprika).

**Error-type eligibility.** GPT-5.6 Terra returns the content types that could be written into each group from its captions: preparation, measurement, timing, temperature, technique (CaptainCook4D); modification, slip, correction (EgoPER). Omission, order, and addition need no judgment.

**Error sampling.** Each video yields one normal copy and six error variants with $\rho \in \{0.25, 0.25, 0.5, 0.5, 0.75, 0.75\}$. A video with $N$ action groups gets a budget $B = \mathrm{round}(\rho N)$, spent in rounds in which every type with a candidate draws one uniformly; order errors and corrections cost 2. Order errors swap two groups ordered by the task graph. A CaptainCook4D group may carry several types, an EgoPER group at most one.

**Error writing.** Omissions and order errors are applied by code. GPT-5.6 Terra writes the other errors of a variant in one call, given the procedural reference, the grouped captions, and the error list (Appendix I.6). A changed amount, duration, or setting must be at most half or at least twice what the reference allows. Examples:

- **Modification** (other tool or ingredient): "sprinkles cinnamon on the chocolate spread" → "sprinkles salt on the Nutella".
- **Slip** (goal not achieved): "spreads the Nutella on the tortilla" → "... onto the red cutting board".
- **Correction** (slip plus a new corrective action): the filter is placed on the table, then "picks up the paper filter from the table and presses it inside the metal coffee dripper".
- **Addition** (new action outside the recipe): "drizzles ketchup over the quesadilla halves".
- **Preparation**: "... sweet and sour sauce with green mold floating inside".
- **Measurement**: "scoops white sugar from a bag with a blue measuring spoon" → "... with the 1/4-cup measuring cup".
- **Timing**: "sets the timer to 4:00" → "8:00" (the ramen reference allows 3:00 or 4:00).
- **Temperature**: "... setting the power level to PL 1".
- **Technique**: "pours the 1/5 teaspoon of ground cinnamon into a black mug" → "... onto the rim of a black mug".
- **Omission / missing step**: the step's captions are removed.

**Teacher answers.** The synthesis record has labels but no explanations, so GPT-5.6 Terra writes the target answer. It receives the detector prompt and an answer key (verdict and types per block, the error lines with their pre-edit text, each omitted step with its insertion time), and explains each error and omission in its own words. An answer is kept only if it reproduces the key and does not mention the key or the edits. The answer key of the EgoPER example in §C.4:

```
Step 1: normal
Step 2: error, slip -- shown by the line [00:21 - 00:34] (was: "the person spreads the
    Nutella on the tortilla with the knife" -> now: "the person spreads the Nutella onto the
     red cutting board with the knife")
Step 3: error, modification -- shown by the line [00:35 - 00:41] (was: "the person sprinkles
    cinnamon on the chocolate spread" -> now: "the person sprinkles salt on the Nutella on
    the red cutting board")
Step 4: normal
Step 5: normal
Step 6: error, addition -- shown by the line [00:57 - 01:00] (a new action: "the person
    drizzles ketchup over the quesadilla halves")
Omitted steps: A4 (InsertAt 00:34, right after Step 2 (A3), the last completed step that must
     come before it)
```

**SFT examples.** The input is the detector prompt: the procedural reference, the variant's captions as step blocks (one per ground-truth group, labelled with its action id; background lines and added actions form background blocks), the error types, and the output format. The target is the teacher's answer; an omitted step's `InsertAt` is $t^*$ (Appendix F.2) on the variant's own groups. The step blocks of the EgoPER example:

```
Step 1 [00:00 - 00:20] A2
  [00:00 - 00:13] the person takes a tortilla from a bag and places it on a red cutting board
  [00:14 - 00:20] the person opens a jar of Nutella and scoops some with a knife
Step 2 [00:21 - 00:34] A3
  [00:21 - 00:34] the person spreads the Nutella onto the red cutting board with the knife
Step 3 [00:35 - 00:41] A5
  [00:35 - 00:41] the person sprinkles salt on the Nutella on the red cutting board
Step 4 [00:42 - 00:49] A6
  [00:42 - 00:49] the person folds the tortilla in half
Step 5 [00:50 - 00:56] A7
  [00:50 - 00:56] the person cuts the folded tortilla in half with a knife
Step 6 [00:57 - 01:04] background
  [00:57 - 01:00] the person drizzles ketchup over the quesadilla halves
  [01:01 - 01:04] the person places the two halves of the quesadilla on a plate
```

and the structure of its target (the error blocks and the final block are shown in §C.4):

```
<step1>
Verdict: normal
</step1>
<step2>
Verdict: error
ErrorType: slip
Explanation: A3 requires spreading Nutella over the tortilla, but the caption says the person
      "spreads the Nutella onto the red cutting board with the knife."
</step2>
...
<final>
OmittedActions: A4 Place 5 banana slices on the chocolate-covered tortilla
InsertAt: 00:34
Explanation: ...
</final>
```

### C.2 Statistics

Tab. 11 summarizes the SFT data.

Table 11: Synthetic SFT data. Error counts are per error event; an order error moves two steps but counts once.

| | EgoPER | CaptainCook4D |
|---|---|---|
| Normal training videos | 186 | 32 |
| Examples (normal + error) | 1,302 (186 + 1,116) | 224 (32 + 192) |
| Errors | 5,653 | 943 |
| Errors per error variant | 5.1 (1–15) | 4.9 (1–10) |
| Step blocks per example | 15.3 | 17.3 |
| Tokens per example (mean / max) | 4,548 / 7,351 | 6,405 / 8,476 |
| *Errors by type* | | |
| EgoPER | slip 1,174, modification 1,162, addition 1,159, omission 1,143, correction 1,015 | |
| CaptainCook4D | preparation 176, missing step 172, technique 170, measurement 163, order 151, timing 106, temperature 5 | |

Tab. 12 breaks the variants down by $\rho$; errors stay below $B$ because some types cost two and types can run out of candidates.

### C.3 Procedural References

Below is the generated reference for EgoPER Quesadilla; the other nine follow the same format. Step ids are the dataset's action ids.

Table 12: Error variants by error density $\rho$. $N$: action groups per video; $B$: error budget; errors: error events per variant; erroneous groups: action groups that carry at least one error, and their share of $N$; removed: groups removed as omissions (included in erroneous groups). All values are means.

| Dataset | $\rho$ | Variants | $N$ | $B$ | Errors | Erroneous groups | Removed |
|---|---|---|---|---|---|---|---|
| EgoPER | 0.25 | 372 | 11.9 | 2.9 | 2.55 | 2.0 (17%) | 0.5 |
| | 0.5 | 372 | 11.9 | 6.0 | 5.05 | 4.0 (34%) | 1.0 |
| | 0.75 | 372 | 11.9 | 9.0 | 7.60 | 6.0 (51%) | 1.5 |
| CaptainCook4D | 0.25 | 64 | 11.1 | 3.0 | 2.66 | 2.9 (26%) | 0.5 |
| | 0.5 | 64 | 11.1 | 6.0 | 5.09 | 5.1 (46%) | 0.9 |
| | 0.75 | 64 | 11.1 | 8.1 | 6.98 | 6.8 (61%) | 1.2 |

**EgoPER, Quesadilla.**

```
Dish Name: Quesadilla

Ingredients / Materials:
- Tortilla
- Chocolate spread or Nutella
- Banana slices
- Cinnamon
- Knife
- Cutting board
- White plate

Core Procedure (with the sub-actions each step contains):
A1. Take one tortilla from its package and lay it flat.
  - Contains: open the tortilla package; take one tortilla; place the tortilla flat
  - Extent: Usually 1-2 lines, about 5-10 s; begins by opening or taking a tortilla from a
      bag or package and ends with the tortilla laid flat, with occasional camera movement
      or ingredient display inside its time span.

A2. Open the chocolate-spread jar and scoop out some spread.
  - Contains: open the jar; scoop chocolate spread from the jar
  - Extent: Usually 1-2 lines, about 8-15 s; begins by getting or opening the jar and ends
      with spread scooped out, with occasional tortilla positioning or early spreading
      inside its time span.

A3. Spread the chocolate spread over the tortilla with a knife.
  - Contains: spread chocolate spread on the tortilla; continue spreading the chocolate
      spread
  - Extent: Usually 1-2 lines, about 13-22 s; begins with the main spreading action and ends
      with the tortilla covered with spread, sometimes including scooping more spread or
      closing the jar.

A5. Sprinkle cinnamon over the chocolate spread.
  - Contains: pick up a spice shaker; sprinkle cinnamon onto the tortilla; sprinkle cinnamon
      again (repeated); set the knife down
  - Extent: Usually 1-2 lines, about 7-10 s; begins by picking up the seasoning or setting
      down the knife and ends with cinnamon on the tortilla, with occasional wiping hands or
       looking at the tortilla inside its time span.

A4. Place 5 banana slices on the chocolate-covered tortilla.
  - Contains: reach for banana slices; take banana slices; place banana slices on the
      tortilla
  - Extent: Usually 1-2 lines, about 8-12 s; begins by getting banana slices from their
      container and ends with them on the chocolate spread, sometimes including setting the
      seasoning down.

A6. Fold the tortilla in half over the filling and press it closed.
  - Contains: adjust the tortilla; fold the tortilla in half; press the folded tortilla;
      press the edges together
  - Extent: Usually 1-2 lines, about 5-8 s; usually begins with folding or briefly adjusting
      the tortilla and ends with a closed half-moon quesadilla, with occasional looking or
      waiting immediately before folding.

A7. Cut the folded quesadilla in half with a knife.
  - Contains: press down on the folded tortilla; cut the folded tortilla in half
  - Extent: Usually 1-2 lines, about 7-12 s; usually begins with the folded quesadilla
      resting on the cutting board or being pressed and ends with two cut pieces, with
      occasional waiting before cutting.

A8. Transfer the two quesadilla halves to a white plate.
  - Contains: pick up the two halves; place the two halves on a white plate; arrange the two
      halves
```

```
  - Extent: Usually 1-2 lines, about 3-6 s; usually begins with the cut pieces on the cutting
       board or being picked up and ends with the two halves plated, though some executions
      stop after picking up or arranging the pieces and plate transfer follows immediately
      afterward.

Background (time in which no step is being carried out):
- Pan across the table, ingredients, counter, wall, or whiteboard -- usually before A1 or
    after A8; this is background display activity rather than preparing the tortilla.
- Open the chocolate-spread jar -- sometimes directly between A1 and A2; opening belongs to
    A2 when it is paired with scooping spread, but is background when it occurs separately
    before A2 begins.
- Adjust the tortilla on the cutting board -- usually between A1 and A2 or between A4/A5 and
    A6; adjustment belongs inside A1, A2, or A6 when it immediately supports placing,
    spreading, or folding, but is background when it is a brief standalone repositioning.
- Look at the tortilla or cut quesadilla -- usually between A3 and A5, before A6, or after A7
    ; this is background inspection rather than filling, folding, or cutting.
- Place the quesadilla halves on a plate -- usually immediately after A8; plating belongs to
    A8 when it follows picking up the halves within the same step, but is background when A8
     ends with picking up, separating, or arranging the pieces.
- Let the plated quesadilla sit or adjust its position on the plate -- usually after A8; this
     is presentation activity after serving.

Ordering Constraints:
- "Opening and scooping the chocolate spread" must be done before "spreading the chocolate
    spread on the tortilla".
- "Spreading the chocolate spread on the tortilla" must be done before "adding the banana
    topping".
- "Spreading the chocolate spread on the tortilla" must be done before "sprinkling cinnamon".
- "Adding the banana topping" must be done before "folding the tortilla over the filling".
- "Sprinkling cinnamon" must be done before "folding the tortilla over the filling".
- "Folding the tortilla over the filling" must be done before "cutting the folded quesadilla".

- "Cutting the folded quesadilla" must be done before "transferring the quesadilla halves to
    a plate".
- Steps not linked by a constraint above may be performed in any order.

Allowed Observational Variants:
- Tortilla packaging: remove the tortilla from a plastic bag or from a tortilla package; the
    tortilla may already be set out.
- Chocolate spread type: Nutella or generic chocolate spread.
- Jar opening method: open the jar with hands or pry it open with a knife.
- Spread handling: scoop the spread before spreading, scoop more spread while spreading, or
    close and set aside the jar during spreading.
- Spread coverage: cover the center, about half, most, or nearly all of the tortilla surface.
- Banana topping form: fresh banana slices, banana chips, or dried banana pieces.
- Banana quantity: 3, 4, or 5 slices.
- Alternative topping type: banana pieces, shredded cheese, or cookies.
- Topping order: add the topping before cinnamon or sprinkle cinnamon before adding the
    topping.
- Seasoning type: cinnamon or a brown spice/seasoning.
- Seasoning container: shaker, spice jar, small container, or grinder.
- Filling closure: fold the tortilla into a half-moon with or without pressing the surface or
     edges closed.
- Cutting result: cut into two halves, two triangular pieces, or triangular wedges.
- Serving action: place the pieces directly on the plate, or first pick up, separate, or
    arrange the pieces before plating.
```

### C.4 SYNTHESIS EXAMPLES

One error variant per dataset: the changed caption lines and the SFT target of the affected blocks (unchanged lines and normal blocks omitted).

**EgoPER, Quesadilla (`quesadilla_u1_a5_normal_029`, $\rho = 0.75$).** Sampled errors: slip (A3), modification (A5), omission (A4), addition.

```
[00:21-00:34] A3 the person spreads the Nutella on the tortilla with the knife
       -> the person spreads the Nutella onto the red cutting board with the knife
[00:35-00:41] A5 the person sprinkles cinnamon on the chocolate spread
       -> the person sprinkles salt on the Nutella on the red cutting board
[00:42-00:50] A4 the person places 5 banana slices on the chocolate spread
       -> (removed)
new line the person drizzles ketchup over the quesadilla halves
```

```
<step2>
Verdict: error
```

```
ErrorType: slip
Explanation: A3 requires spreading Nutella over the tortilla, but the caption says the person
     "spreads the Nutella onto the red cutting board with the knife."
</step2>
<step3>
Verdict: error
ErrorType: modification
Explanation: A5 requires sprinkling cinnamon or a brown seasoning over the chocolate spread,
    but the caption says the person "sprinkles salt on the Nutella on the red cutting board."

</step3>
<step6>
Verdict: error
ErrorType: addition
Explanation: The person "drizzles ketchup over the quesadilla halves," which is not part of
    any reference step or allowed variant.
</step6>
<final>
OmittedActions: A4 Place 5 banana slices on the chocolate-covered tortilla
InsertAt: 00:34
Explanation: A4 requires taking banana slices and placing them on the chocolate-covered
    tortilla, but no block shows banana slices being added; it belongs right after Step 2 (
    A3), spreading the Nutella onto the red cutting board with the knife.
</final>
```

**CaptainCook4D, Spiced Hot Chocolate (`spicedhotchocolate_normal_8_16`, $\rho = 0.5$).** Sampled errors: preparation (A7), technique (A2), measurement (A5), missing step (A1).

```
[01:08-01:12] A7 closes the microwave door and sets the timer to 1:00
       -> closes the microwave door with a metal spoon inside the black mug and sets the
            timer to 1:00
[03:23-03:28] A2 pours the 1/5 teaspoon of ground cinnamon into a black mug containing a
    white liquid
       -> pours the ground cinnamon onto the rim of a black mug containing a white liquid
[03:52-04:46] A5 picks up a blue measuring spoon ... scoops white sugar ... adds the white
    sugar to a mug of hot chocolate
       -> picks up a 1/4-cup measuring cup ... scoops white sugar with the 1/4-cup measuring
            cup ... adds 1/4 cup of white sugar to a mug of hot chocolate
[06:20-07:32] A1 (all 10 lines removed)
```

```
<step3>
Verdict: error
ErrorType: preparation
Explanation: The milk should be heated in the mug without an improper utensil left inside,
    but the person "closes the microwave door with a metal spoon inside the black mug."
</step3>
<step5>
Verdict: error
ErrorType: technique
Explanation: The cinnamon should be poured into the hot milk, but the person "pours the
    ground cinnamon onto the rim of a black mug."
</step5>
<step7>
Verdict: error
ErrorType: measurement
Explanation: The procedure requires a spoonful of sugar, but the person uses a "1/4-cup
    measuring cup" and "adds 1/4 cup of white sugar to a mug of hot chocolate."
</step7>
<final>
OmittedActions: A1 Microwave the stirred hot chocolate for 1 minute
InsertAt: 06:19
Explanation: A1 requires placing the stirred mug in the microwave, heating it for 1 minute,
    and removing it, but no block shows this final microwave heating; it belongs right after
     Step 8 (A3), stirring the hot chocolate with a metal spoon.
</final>
```

# D Evaluation Metrics

Rows we score ourselves use the released metric functions of EgoPER Lee et al. (2024b) and AXG-Reasoner Lee & Elhamifar (2026) on the datasets' annotations; detection on ground-truth segments re-implements the metric of AXG-Reasoner. Values copied from papers follow those papers (Appendix E). A ground-truth segment is erroneous when it has any error type, so CaptainCook4D

steps tagged only with an order error count as errors. Frame-level metrics use 10 fps labels; a caption line $[s, e]$ covers $[s, e+1)$ seconds. "All" is the mean over the five tasks.

**TAS.** The official EgoPER scorer: background is a class and only EgoPER's $-1$ label is excluded; Edit and F1@0.5 are averaged over videos and Acc is pooled over frames.

**Error detection F1@0.5.** AXG-Reasoner's protocol: flagged blocks give a binary frame sequence, normal and error runs are matched to the ground truth at IoU 0.5 with counts pooled over a task's videos, and F1@0.5 is the mean of the normal (N) and error (E) F1.

**Detection on ground-truth segments.** Each annotated segment gets one decision; the normal and error F1 are computed per video over segments, averaged over videos, and F1 is their mean. For TGPED, each annotated segment is one step block holding the caption lines that overlap it by at least 1 s.

**EDA and AUC.** The released EgoPER code. EDA is the share of ground-truth segments (split at action-label changes, background included) whose prediction is correct, a segment being predicted erroneous when any of its frames is flagged. AUC sweeps 41 thresholds over frame-level true- and false-positive rates. Our detector has no score and gives a single operating point; AEM on CaptainCook4D uses its own 41 thresholds from 0 to 1, and its EDA is their mean.

**Omission localization F1@$\tau$.** Predictions are matched one-to-one to ground-truth omissions by action id; a match is a true positive if $|\hat{t} - t^*| \leq \frac{\tau}{100} L_v$, with $L_v$ the video length ($t^*$ and $\hat{t}$ in Appendix F). F1 is computed per task over all its test videos, for $\tau \in \{1, 2.5, 5, 10\}\%$.

### D.1 Pairwise Explanation Comparison

Both methods run on ground-truth segments. For each ground-truth error segment, a method's explanation is that of its predicted segment overlapping it most (for AXG-Reasoner, the concatenated reasoning of its flagged sub-actions). On segments that both methods flag, GPT-5.6 Terra receives the official step list, the two explanations, and a question built from the segment's annotated step and error description for its error type; it must pick one, and each pair is judged twice with the order swapped. Tab. 6 reports TGPED's win rate over these segments. Before judging, action ids and block numbers in TGPED's explanations are replaced by step names. CaptainCook4D types map to the EgoPER questions (preparation, measurement, timing, temperature $\rightarrow$ modification; technique $\rightarrow$ slip; order has its own); missing steps are not scored. Prompts are in Appendix I.9.

## E Baselines

Tab. 13 gives the source of every baseline number. When predictions, checkpoints, or code are released, we score the method with our evaluation code and ground truth; otherwise we copy the paper's value.

Table 13: Source of each baseline result. P: reported in a paper; R: released predictions re-scored; C: released checkpoints run; T: trained with the released code; X: released code run; S: action segmentation released by AXG-Reasoner Lee & Elhamifar (2026). (E): EgoPER only. –: not reported.

| Method | TAS Tab. 1 | Det. F1 Tab. 2 | EDA / AUC Tab. 3 | GT seg. Tab. 4 | Expl. Tab. 5, 6 | Omission Tab. 7 |
|---|---|---|---|---|---|---|
| EgoPED Lee et al. (2024b) | T (E) | P | P (E) | – | – | S |
| AMNAR Huang et al. (2025b) | R (E) | – | P (E) | – | – | R (E) |
| State-Change Kung et al. (2025) | – | – | P (E) | – | – | – |
| AEM Guo et al. (2026) | – | – | P (E), C | – | – | – |
| GTG2Vid Lee & Elhamifar (2025) | C | C | – | – | – | C |
| AXG-Reasoner Lee & Elhamifar (2026) | – | X | – | X | X | – |

**EgoPED.** It releases no model or predictions; we train it (two-stage, I3D features) on EgoPER with its released code for TAS. For omission localization we use its segmentation released by AXG-Reasoner, with action ids mapped by step name.

**AMNAR.** We re-score its released segmentation for TAS and omission localization.

**AEM.** We run its released per-recipe checkpoints on our 40 CaptainCook4D test videos for EDA and AUC; its paper averages over all 24 recipes.

**GTG2Vid.** We run the official checkpoints and use its segmentation for TAS, its error output for detection, and its node-dropping alignment for omission localization, placed by the $t^*$ rule (Appendix F).

**AXG-Reasoner.** We run the released code in its paper's configuration ($K$=5 clusters, 8 frames, GTG2Vid segments) with Qwen3.6-27B instead of Qwen2.5-VL-32B; ground-truth-segment results use its ground-truth mode and its explanation judge (Qwen2.5-32B-Instruct).

# F Omission Ground Truth & Localization Time

Omission localization (Tab. 7) needs the omitted steps and the time each should have been performed. Neither dataset annotates this time, so we derive a reference time $t^*$ from the task graph; the metric is given in Appendix D.

## F.1 Ground-Truth Omitted Steps

EgoPER has no segment for an omitted step, so every task-graph action without a ground-truth segment in the video is omitted (293 in the test set). CaptainCook4D annotates omissions directly; we use its *Missing Step* labels (40 in the test set).

## F.2 Reference Time $t^*$ from the Task Graph

$t^*$ is the end time of the omitted step's nearest performed predecessor in the task graph: we walk the graph backward and stop each branch at the first performed action. Several branches take the latest end; a repeated predecessor takes its last occurrence before the steps that follow the omitted one; with no performed predecessor, $t^* = 0$.

## F.3 Predicted Time $\hat{t}$

**Baselines.** All baselines use the official task graph. For EgoPED, AMNAR, and GTG2Vid, a task-graph action absent from the method's segments is omitted (for GTG2Vid, we rerun its node-dropping alignment, since its released output keeps only the node-preserving one), and $\hat{t}$ applies the $t^*$ rule to the method's own segments. Tab. 14 also reports GTG2Vid with $\hat{t}$ at the frame where each node is dropped; the omitted steps are the same.

Table 14: Omission localization F1 (%) of GTG2Vid. (a) is the row of Tab. 7.

| | EgoPER | | | | CaptainCook4D | | | |
|---|---|---|---|---|---|---|---|---|
| $\tau$ (%) | 1 | 2.5 | 5 | 10 | 1 | 2.5 | 5 | 10 |
| (a) $t^*$ rule on its segments | 26.8 | 32.1 | 38.3 | 46.1 | 11.4 | 17.6 | 19.8 | 22.4 |
| (b) dropped-node frame | 17.4 | 22.1 | 26.4 | 35.2 | 10.5 | 19.2 | 20.2 | 23.3 |

**Ours.** The detector never sees the task graph at test time. It lists each omitted step by its action id with an `InsertAt` time, which is $\hat{t}$, and an explanation naming the step block it belongs after (§C.4). In the training targets, `InsertAt` is $t^*$ computed on the synthesized video's own action groups, so the task graph is used only to build training targets.

Table 15: Detection rate (%) per error type (number of segments in parentheses). *type*: correctly typed share of detected segments. Omission: F1@10% (Tab. 7). Normal flagged: lower is better.

**EgoPER**

| Method | Modification (215) | Slip (143) | Addition (104) | Correction (42) | Omission | Normal flagged (1,783) |
|---|---|---|---|---|---|---|
| GTG2Vid | 47.9 | 44.1 | 73.1 | 57.1 | 35.2 | 19.0 |
| AXG-Reasoner | 50.7 | 58.7 | 61.5 | 45.2 | – | 36.5 |
| TGPED (ZS) | 65.6 | 48.3 | 67.3 | 42.9 | 33.2 | 32.0 |
| *type* | 68.1 | 69.6 | 47.1 | 5.6 | | |
| **TGPED** | 63.7 | 40.6 | 76.9 | 35.7 | 44.0 | 15.8 |
| *type* | 81.8 | 89.7 | 58.8 | 6.7 | | |

**CaptainCook4D**

| Method | Order (101) | Prep. (78) | Tech. (71) | Meas. (53) | Timing (38) | Temp. (7) | Missing step | Normal flagged (157) |
|---|---|---|---|---|---|---|---|---|
| GTG2Vid | 65.3 | 57.7 | 47.9 | 66.0 | 68.4 | 100.0 | 23.3 | 46.5 |
| AXG-Reasoner | 87.1 | 92.3 | 83.1 | 96.2 | 94.7 | 100.0 | – | 80.9 |
| TGPED (ZS) | 64.4 | 56.4 | 47.9 | 73.6 | 50.0 | 57.1 | 35.0 | 41.4 |
| *type* | 60.0 | 36.4 | 32.4 | 12.8 | 63.2 | 25.0 | | |
| **TGPED** | 75.2 | 55.1 | 42.3 | 67.9 | 81.6 | 71.4 | 47.4 | 35.0 |
| *type* | 52.6 | 27.9 | 33.3 | 27.8 | 80.6 | 40.0 | | |

## G Different Error Type Performance

Tab. 15 gives the detection rate per annotated error type: the share of ground-truth error segments with any flagged frame, over all test videos (a CaptainCook4D step counts once per tag). The share of normal action segments flagged by the same test is the corresponding false alarm rate. For our detectors, *type* is the share of detected segments whose error type is correct.

AXG-Reasoner's high detection rates on CaptainCook4D come with flagging 81% of the normal segments; TGPED flags the fewest normal segments on both datasets. Fine-tuning halves the false alarms on EgoPER and improves typing of modifications and slips, at the cost of detecting fewer slips and corrections. Corrections are rarely typed correctly: TGPED calls 13 of the 15 it detects a slip.

## H Qualitative Examples

TGPED on three test videos: a mostly successful one per dataset and a failure. Each listing shows PGTS step blocks (span and action id), captions, the detector output, and the ground truth (GT); blocks normal in both are omitted.

**EgoPER, Tea (`tea_u1_a1_error_021`, 10 blocks).** Two of three errors are found with correct types and explanations, and the omission is placed 4 s from $t^*$; the sweetener error is missed although the caption states it.

```
Step 3 [00:35 - 01:18] A3 GT: A3, slip (drop tea bag)
  [00:48 - 00:52] the person drops the tea bag on the floor
  [00:53 - 00:55] the person picks up the tea bag from the floor
  ...
  -> error, slip: The tea bag should be placed in the mug after taking it from its wrapper,
       but the person "drops the tea bag on the floor" before picking it up and putting it in
       the cup.

Step 8 [01:48 - 01:54] A8 GT: A8, modification (sugar instead of honey)
  [01:48 - 01:54] opens a pink sweetener packet and pours the contents into the blue mug
  -> normal (missed)

Step 9 [01:55 - 02:02] A9 GT: A9, modification (stir with knife)
  [01:55 - 01:58] the person reaches for a knife on a white plate
  [01:59 - 02:02] stirs the contents of the blue mug with the knife
  -> error, modification: The reference requires stirring the honey into the tea with a spoon,
        but the person reaches for and uses a knife instead.

Final block GT: A4 omitted, t* = 00:30
  OmittedActions: A4 Check the temperature of the water in the kettle with a thermometer
  InsertAt: 00:34
```

```
  Explanation: A4 requires inserting a thermometer into the kettle and checking the water
       temperature, but no block shows this action; it belongs right after Step 2 (A2), the
       person closes the kettle.
```

**CaptainCook4D, Spiced Hot Chocolate (`spicedhotchocolate_error_8_50`, 8 blocks).** Both order errors and both omissions are found (placed 3 s and 65 s after $t^*$). The first heating is flagged as a timing error, which its caption supports (2:35 for a 1-minute step) although it is annotated as an order error. The chocolate spill is captioned during the next step (03:21, after A8 ends at 03:06), so it is reported on the wrong block.

```
Step 2 [00:08 - 01:07] A5 GT: A5, order
  [00:40 - 00:47] scoops powdered sugar from the bag with the measuring spoon
  [00:48 - 00:53] pours the powdered sugar from the measuring spoon into the grey mug
  ...
  -> error, order: The person adds powdered sugar to the mug before Step 3 pours milk into it,
       but A6 must be completed before A5.

Step 3 [01:08 - 01:29] A6 GT: A6, order
  [01:15 - 01:23] takes a jug of milk from the refrigerator and pours it into the mug
  ...
  -> error, order: The person pours milk into the mug after Step 2 has already added sugar,
       but A6 must occur before A5.

Step 4 [01:30 - 02:43] A7 GT: A7, order
  [01:46 - 02:36] the microwave runs, the display counts down from 2:35
  ...
  -> error, timing: The first microwave heating uses a "2:35" countdown rather than the
       required one-minute heating time.

Step 5 [02:44 - 02:59] A8 GT: A8, technique (spill chocolate)
  [02:44 - 02:50] opens a cabinet and takes out a bag of chocolate chips
  [02:51 - 02:59] opens the bag of chocolate chips
  -> normal

Step 7 [03:04 - 04:11] A1 GT: A1, normal
  [03:21 - 03:25] pours chocolate chips from the bag into a clear plastic ziplock bag, some
       chips spill onto the counter
  [03:38 - 03:42] picks up the spilled chocolate chips from the counter with their hands
  ...
  -> error, technique, preparation: Instead of adding chocolate chips to the mug, the person
       transfers them into a ziplock bag and discards spilled chips, so the final microwave
       heating occurs with only milk in the mug.

Final block GT: A2, A3 omitted, t* = 02:40, 03:06
  OmittedActions: A2 Add 1 teaspoon of ground cinnamon to the hot milk; A3 Stir the hot
       chocolate with a spoon until the ingredients are mixed
  InsertAt: 02:43; 04:11
  Explanation: A2 requires taking out cinnamon, measuring it, and adding it to the hot milk,
       but no block shows cinnamon being handled or added; it belongs right after Step 4 (A7),
        heating the milk-filled mug in the microwave. A3 requires stirring the hot chocolate
       with a spoon until the ingredients are mixed, but no block shows stirring; it belongs
       right after Step 7 (A1), microwaving the stirred hot chocolate.
```

**Failure: EgoPER, Coffee (`coffee_u1_a2_error_019`, 16 blocks).** All three errors and the omission are missed, and one normal step is flagged, showing four failure sources: (i) *missing evidence*: the captions omit the water amount (Step 3) and show the early dripper removal as camera motion (Step 13); (ii) *detector miss*: the captioned bean spill is judged normal (Step 6); (iii) *captioning error*: the thermometer is captioned as a knife, causing a false alarm (Step 9); (iv) *segmentation*: PGTS assigns the omitted first pour (A14) to the main pour.

```
Step 3 [00:41 - 00:50] A1 GT: A1, slip (incorrect amount of water)
  [00:41 - 00:50] the person pours water from a plastic container into a glass measuring cup
  -> normal

Step 6 [01:30 - 01:34] A10 GT: A10, slip (spill coffee beans)
  [01:30 - 01:34] pours coffee beans from a white bowl into a black coffee grinder, some beans
        spill onto the table
  -> normal

Step 9 [02:14 - 02:37] A13 GT: A13, normal
  [02:14 - 02:23] places the coffee grinder on the table and picks up a red kitchen knife
  [02:24 - 02:30] inserts the red kitchen knife into a black kettle
  ...
```

```
  -> error, modification: The temperature-checking step requires a digital thermometer, but
       the person "picks up a red kitchen knife" and inserts it into the kettle instead.

Step 10 [02:38 - 02:59] A14 GT: A15 (A14 omitted, t* = 02:29)
  [02:41 - 02:59] pours hot water from the black kettle into the coffee filter over the ground
       coffee
  ...
  -> normal

Step 13 [03:22 - 03:32] A3 GT: A3, slip (remove dripper before drained)
  [03:22 - 03:24] the person reaches for the coffee filter
  [03:25 - 03:29] the camera moves rapidly, blurring the view of the table and floor
  [03:30 - 03:32] places the used coffee filter and grounds into a trash bin
  -> normal

Final block GT: A14 omitted
  OmittedActions: none
```

# I PROMPTS

We list every prompt used by our pipeline and evaluation, verbatim. Text in braces, such as {dish} or {lines}, is filled in at run time.

## I.1 DENSE VIDEO CAPTIONING

Qwen3.6-27B, one call per 90 s window; the window's frames precede the prompt.

```
You are given {n_frames} frames from ONE clip of a procedural task video, in strict temporal
     order.

Context:
- The video shows a person preparing {dish}.
- This clip runs from {start} to {end} ({seconds} seconds). It may show one step of the task
     execution, part of a step, or only waiting or moving around between steps.
- Every frame is labelled with its timestamp in the video.

Your task is to produce a detailed timeline of this clip.

Output format:
[MM:SS - MM:SS] description
...

Requirements:
1. Divide the clip into consecutive segments that cover the visible activity. Cut a new
     segment
   whenever the action, the object being handled, or the tool changes. Do NOT merge several
   distinct actions into one segment and do NOT summarize; use as many segments as the visible
   activity requires.
2. Timestamps are VIDEO timestamps read from the frame labels: the first segment starts at
   {start} and the last segment ends at {end}.
3. Each segment must have a start time and an end time and describe the main action in that
     interval.
4. Descriptions must:
   - be one sentence stating what is done, to which object, and with which tool.
   - before writing a segment, check four things and write whichever is visible:
     (a) how much or how many, and with which utensil: count the pieces, cupfuls, pours or
        repetitions you can see ("adds two cups of lentils with a measuring cup");
     (b) any digits shown on a timer, scale, dial or display, read exactly ("sets the
        kitchen timer to 15:00"); if the display is not legible, say so ("timer not legible");
     (c) anything spilled, dropped, splashed, left behind or done incompletely ("some lentils
        fall onto the counter", "a few slices stay on the cutting board");
     (d) the result of the action when it can be seen ("the water is boiling", "the carrot
        is cut into uneven chunks").
   - follow temporal order strictly
5. Ensure no overlapping time ranges and no gaps between segments.
6. Use the frame timestamps to estimate segment boundaries as accurately as possible.

7. Only describe manipulations where the person actively interacts with objects (pouring,
     placing,
cutting). Camera motion and scene views are states, not actions.

Quantity evidence requirement:
8. Whenever quantities are involved, report the quantity evidence you can actually see:
   - counts of discrete items: "places 3 carrots on the board", "adds 2 bay leaves"
   - repetitions of an action: "sprinkles salt into the pot twice", "pours water in three goes
        "
```

```
   - the measuring utensil used: "adds a level teaspoon of salt", "measures with a measuring
       cup"
   - approximate visible amounts: "fills the pot about halfway", "covers the lentils with
       water"
   - a scale, timer or dial reading ONLY if the digits are legible: "sets the kitchen timer to
        15:00"
   Report counts, repetitions and the utensil even when no numeric reading is legible.
   Never invent a reading you cannot see, and never state a quantity merely because the recipe
   expects it -- describe what happened in this clip.
9. Every concrete fact goes into the segment in which it is visible; there is no separate
     summary.
   Besides (a)-(d) above, name the ingredient or food handled and the container or vessel, and
        state
   how long an action visibly went on and any heat or power setting, whenever these are
       visible.
   Include ONLY facts that are visible in these frames. Do not write about things that are not
   visible and do not guess.

Output rules:
- Output ONLY the timeline lines
- Do NOT include explanations or extra text

Example output (a different dish, showing the level of detail expected):
[00:00 - 00:06] takes a pot from a lower cabinet and places it on the stove
[00:07 - 00:15] pours water from a jug into the pot, filling it about halfway
[00:16 - 00:19] the jug is set down, the person looks at the stove
[00:20 - 00:31] adds two cups of red lentils with a measuring cup, one cup at a time
[00:32 - 00:36] turns the stove dial to the medium setting
[00:37 - 00:49] stirs the lentils with a wooden spoon, some water splashes onto the stove
[00:50 - 01:05] the pot heats on the stove, the person stands and watches
[01:06 - 01:12] sets the kitchen timer to 15:00
[01:13 - 01:25] chops a carrot into round slices with a knife on a cutting board
[01:26 - 01:34] chops a second carrot, a few slices roll onto the counter
[01:35 - 01:43] scrapes the carrot slices into the pot with the knife
[01:44 - 01:52] sprinkles salt into the pot in two pinches

Grounding rules:
- The example above is a different dish. Never reuse its sentences or its details; every
     quantity,
  reading, spill or result you write must be visible in THESE frames. If you cannot see it,
       write the
  plain action without it.
- A timer, scale or dial reading is written only when the digits are legible in a frame,
     exactly as
  shown; otherwise write "display not legible".
- Keep the segmentation: do not merge separate actions into one segment to make room for
     detail;
  a repeated action (a second scoop, a second sprinkle) is its own segment.
```

### I.2 Caption Refinement

GPT-5.6 Terra, one call per ground-truth action segment of a training video.

```
Below are caption lines from an automatic dense video caption of a person carrying out a
     procedure, and the ground-truth action that the person is performing during these lines.
      The ground truth is correct; the captions were written by a captioning model and may
     misread a detail.

Ground-truth action: {gt_action}
(The action just before it: {prev_action}. The action just after it: {next_action}.)

Caption lines:
{lines}

Read the lines together, in order. Repair only a line that states something the ground-truth
     action rules out. There are two kinds:
1. A number, time, duration, amount, count or setting that differs from the one the action
     states, including a vague amount or count ("a few", "some", "several", "more") where the
      action states one (the action requires a timer of 5 minutes and the line says the timer
      is set to 2:40; the action adds 3 pieces and the line says a few pieces). Replace it
     with a single value written the way the line writes it ("5:00", "3 pieces"); when the
     action gives a range or an approximate value, use one value from it, never the range
     itself.
2. An ingredient, object or result that is a different thing from the one the action involves
      (the action adds salt and the line says sugar), including an item named by a similar-
     looking or related item (a bun or a roll for a bagel, a tray used as the cover the
     action names), and a vague description of the ingredient or object the action names: a
     colour or appearance instead of a name, or a general word such as "sauce", "powder", "
```

```
     seasoning" or "spread" (the action adds paprika and the line says "red powder"), or a
     generic word for the specific container or tool the action names (a container for the
     jar, a pot for the saucepan, a spoon for the ladle). Replace it with the action's name
     for the item.

Do NOT repair, and leave unchanged:
- A different name or description for the same thing: a synonym or a regional or everyday
     name (a mug for a cup, a stovetop for a cooktop), or the same object or place described
     by its shape, by the pieces it was cut into, or by what it rests on.
- A part of the action: an action can be carried out in several parts (several shorter timer
     runs, an amount added in several scoops), so a part is not a conflict just because it is
      smaller than the whole.
- An extra detail the action does not mention: a brand, a colour, an ingredient that was
     already added earlier.
- Fetching, opening, washing, putting away, waiting, looking at a screen, camera movement, or
      any other activity the ground truth does not mention.
- A line that shows the action just before or just after, or any other action than the ground
      truth: its time only overlaps this action's boundary.
When in doubt, leave the line unchanged.

When you repair, replace only the conflicting words and keep every other word exactly as
     written. A detail from the ground truth may be written into a line only if it is
     something that can be seen in the video (an object, an ingredient, a count, a value on a
      display), even if the captioner did not recognise it; never write in a property that
     cannot be seen (a material, "microwave-safe", a fat content, a capacity), and never
     remove a visible detail such as a colour. The repaired lines must still read
     consistently together: if several lines repeat a step more times than the action
     requires (a second, third and fourth piece where the action uses one), do not give them
     all the same number; write the repeats as continuing the same step on the same item ("
     cuts the piece further").

Return one output line for each caption line you repaired, and nothing else:
Line <k>: <repaired caption> || <the conflicting detail you replaced>
If no line needs a repair, return exactly: none
```

### I.3 Procedural Reference

GPT-5.6 Terra, one call per task over all of its normal training videos.

```
You are given dense video captions of several NORMAL demonstrations of the same task: {dish}.
      Every demonstration was carried out correctly.

Each demonstration is divided into blocks, in temporal order. A line "[Action A<id> mm:ss -
     mm:ss]" opens an action block: A<id> is an action id, the time span is when the ground
     truth marks that action, and the caption lines under it are the lines that fall mostly
     inside that span. A line "[Background]" opens a background block: its caption lines fall
      mostly in the ground truth's background (BG) time, when no numbered action is being
     carried out. An action shorter than a caption line may have no line mostly inside it;
     its block then lists the lines that overlap it, which also appear under their own block.
      The same A<id> is the same action in every demonstration. The ids carry no meaning
     beyond identity. The captions were written by an automatic captioner and can misread a
     detail in any single demonstration.

Write one practical recipe for this task that a person could follow, using only what the
     demonstrations show.

Rules:
1. Write exactly ONE Core Procedure step per action id that appears in the demonstrations;
     never split one id into several steps or merge two ids into one step. Order the steps by
      the order in which the actions most commonly occur. Label every step with its action id
      in place of a number, e.g. "A7. Chop the onion into small pieces"; the ids therefore do
      not run in order, and every id is used exactly once.
2. Each step is one concrete imperative instruction describing the main action that the block'
     s caption lines show across the demonstrations: the ingredient, the tool or container,
     the manner and the result. Name things the way most demonstrations show them.
3. Every step whose demonstrations state a quantity, count, time, temperature or setting (a
     timer or display reading, a number of pieces or scoops, a fill level) MUST state it in
     the step itself: among the demonstrations that state a value for this action, use the
     value most of them agree on. The other observed values go to the Allowed Observational
     Variants in addition, never instead. Leave a value out of a step only when no
     demonstration states one. Never take a value from outside knowledge of the dish.
4. A quantity prescribes an amount only, never an implement. If the demonstrations use
     different implements or containers for the same action, the step names none of them and
     the Allowed Observational Variants list them.
5. Directly under each step add a line " - Contains:" listing, separated by semicolons, the
     sub-actions that the caption lines of this action's blocks show, as short verb phrases
     in the captions' own words. List each distinct sub-action once, in its usual order, at
     most 12 items; write a repeated sub-action once with "(repeated)".
```

```
6. Directly under Contains add a line " - Extent:" that first gives the step's usual length
     in the demonstrations -- about how many caption lines and how many seconds its blocks
     span, as a typical value or a narrow range (e.g. "Usually 1-2 lines, about 10 s") -- and
      then states, from this action's blocks, how the step begins and ends: whether its block
      usually opens with getting its ingredient, tool or container ready (taking it from the
     refrigerator, a cabinet or a drawer, opening a package, setting a bowl on the counter)
     or only with the main action; what it usually closes with (putting the item back,
     closing a lid, the finished result); and whether other activity (checking the recipe,
     cleaning, waiting for an appliance) falls inside the step's time span. Report what most
     demonstrations show; when they disagree, say so.
7. Background: describe what the caption lines of the background blocks show across the
     demonstrations -- the kinds of activity, as short verb phrases in the captions' own
     words, and where in the procedure each usually occurs (before the first step, between
     which steps, after the last step). When the same kind of activity also appears inside an
      action's blocks (for example getting an item ready appears both in a step's block and
     in a background block), state what the demonstrations show about which cases belong to a
      step and which are background: which step it belongs to, whether it directly precedes
     or follows that step, and what is different when it is background. Include a kind that
     recurs across demonstrations; leave out one seen only once.
8. Ordering Constraints: list only direct dependencies, where the later step works on the
     result of the earlier one (the egg is whisked after it is cracked into the bowl; the
     grounds go into the filter after the beans are ground), and only when every
     demonstration that contains both steps follows that order. Do not list an order that is
     merely usual. Name both steps by a short description of their action, never by their id.
9. Allowed Observational Variants: list every difference between the demonstrations that a
     correct execution of this task could also show -- substitutions, alternative values,
     optional extra actions, ingredient forms, or anything else that recurs across
     demonstrations -- naming the KIND that varies (not only its colour or brand) and the
     alternatives observed. A reviewer uses this list to avoid flagging a normal execution as
      an error, so be complete. Leave out a value that is an outlier: far from the values the
      other demonstrations show, physically implausible, in a different unit, or most likely
     a misread display.
10. An ingredient, tool or sub-action that appears in only a few demonstrations goes into the
      Allowed Observational Variants, not into the Ingredients / Materials, the steps or
     their Contains lists.

Output exactly these sections, in this order, with the bold labels shown. Keep it concise and
      recipe-like; do not mention videos, captions, demonstrations, blocks, evidence or your
     reasoning (the length in lines on the Extent line is the one exception):

**Dish Name:** {dish}

**Ingredients / Materials:**
- <item>

**Core Procedure (with the sub-actions each step contains):**
A<id>. <instruction>
  - Contains: <sub-action>; <sub-action>
  - Extent: <usual length: about N lines, about N s; how the step usually begins (getting the
       item ready, or the main action) and ends; other activity inside its time span>
A<id>. <instruction>
  - Contains: <sub-action>; <sub-action>
  - Extent: <usual length: about N lines, about N s; how the step usually begins (getting the
       item ready, or the main action) and ends; other activity inside its time span>

**Background (time in which no step is being carried out):**
- <a kind of activity> -- <where in the procedure it usually occurs; if the same activity
     also occurs inside a step: which step it belongs to, whether it directly precedes or
     follows it, and what is different when it is background>

**Ordering Constraints:**
- "<short description of the earlier step>" must be done before "<short description of the
     later step>".
- Steps not linked by a constraint above may be performed in any order.

**Allowed Observational Variants:**
- <a kind of thing that differs between correct executions of this task, with the observed
     alternatives>

Demonstrations:

{demonstrations}
```

### I.4 PGTS

Qwen3.6-27B with thinking, one call per test video.

```
Below is a recipe and the numbered caption lines of one video of a person preparing this dish,
      in temporal order. Mark the step segments: each segment is one recipe step, carried out
      from its first line to its last. Lines outside every step segment are background.

Recipe:
{recipe}

Caption lines:
{lines}

The recipe was written from other recordings of this dish. Its Core Procedure gives each step
      an id (A1, A2, ...), what the step contains, and how it usually begins and ends. Its
     Background section describes what happens between steps.

A step segment runs from the line where the step begins to the line where it ends; every line
      in between is part of it, whatever it shows. Background is only the lines between one
     step's end and the next step's beginning, before the first step, or after the last.
     Steps may occur in a different order than listed, be missing, be repeated later (a new
     segment each time), or be done wrongly; record what the video shows. The captions come
     from an automatic captioner and may misname an object; go by the action.

Return the step segments in order, one per line, and nothing else. Segments must not overlap:
      each line is in at most one segment. If another step is carried out in the middle of a
     step, end the segment before it and start a new segment for the rest after it. What you
     return is one non-overlapping segmentation of the video in temporal order; where two
     steps overlap in the video, mark only the main one on those lines. Do not list
     background lines; every line left out is background.
Lines <first>-<last>: <step id, e.g. A3>
```

### I.5 Error-Type Eligibility

GPT-5.6 Terra, one call per action group of a training video.

**CaptainCook4D.**

```
Below are the caption lines of one segment of a video of a person preparing {dish}. The
     segment is one action of the procedure, carried out correctly. The captions were written
      by an automatic captioner, in temporal order.

Caption lines:
{lines}

Error types:
- preparation: soiled or wrong ingredients are used, or a different tool.
- measurement: a wrongly measured amount of an ingredient is used.
- timing: the action is done for a shorter or longer duration than it should be.
- temperature: a higher or lower heat or power level is set than it should be.
- technique: the required action is performed incorrectly, leading to a wrong outcome.

Decide which error types could be written into this segment. A type is eligible only if at
     least one caption line shows the detail that type would change (the ingredient or tool;
     the amount -- a stated quantity or count, or a measuring spoon, cup or scale in use; the
      duration -- a timer, a displayed time or a stated length of time; the heat or power
     level; the manner or outcome of the action), so that changing a few of its words would
     state the error as a visible, observed fact. A detail that no caption line shows cannot
     be changed, so its type is not eligible.

Return exactly one line:
Eligible: <comma-separated types from preparation, measurement, timing, temperature,
     technique, or none>
```

**EgoPER.**

```
Below are the caption lines of one segment of a video of a person preparing {dish}. The
     segment is one action of the procedure, carried out correctly. The captions were written
      by an automatic captioner, in temporal order.

Caption lines:
{lines}

Error types:
- modification: the action is performed in a different way than it should be, e.g. with a
     different tool (stirring with a knife instead of a spoon) or a different ingredient (
     sugar instead of honey); the outcome does not necessarily change.
- slip: the action is executed in a way that does not achieve its goal, e.g. water is poured
     into a different bowl, or the tortilla is dropped on the floor.
- correction: the action first goes wrong as in a slip, and the person then performs a
     corrective action that mitigates it, e.g. transferring the water back to the right bowl,
      or discarding the dropped tortilla and taking a new one.
```

```
Decide which error types could be written into this segment. A type is eligible only if at
     least one caption line shows the part of the action that type would change (the tool or
     ingredient for modification; the execution or result of the action for slip and
     correction), so that changing a few of its words -- and, for correction, adding the
     corrective action after it -- would state the error as a visible, observed fact. A
     detail that no caption line shows cannot be changed, so its type is not eligible.

Return exactly one line:
Eligible: <comma-separated types from modification, slip, correction, or none>
```

### I.6 Error Writer

GPT-5.6 Terra, one call per error variant. {types} lists the definitions of the dataset's error types.

```
Below is the procedural reference for {dish}, and the dense captions of a video of a person
     following it. The captions were written by an automatic captioner, in temporal order,
     and are grouped by the step of the reference each line shows (<no_step> for lines
     between steps; a step done a second time is labelled like A13 (2)); each line has a
     number n and its duration in seconds. The video shows the procedure carried out
     correctly.

<reference>
{reference}
</reference>

<captions>
{lines}
</captions>

Rewrite the captions so that the video contains the following errors, each in the step and
     lines given:
<errors>
{errors}
</errors>

The error types of this dataset:
<error_types>
{types}
</error_types>

Rules:
1. State each error as a visible, observed fact: what the person uses, how much, for how long,
      at what setting, how they do it, or what happens. Only describe what is seen; never say
      how it should have been done or that something went wrong: no "instead of", "rather
     than", "in place of", "wrong", "mistakenly", "incorrectly", "accidentally", "too much"
     or similar words.
2. Each error must be something a real person preparing this dish could do, with things
     plausibly at hand. An allowed variant listed in the reference is not an error: before
     writing an ingredient, tool, container, amount or setting, check the reference's Allowed
      Observational Variants, and never write one listed there -- not even when it is the
     example given in an error type's definition (if the reference allows sugar as a
     sweetener, using sugar is not an error).
3. A wrong amount, duration or setting must lie clearly outside everything the reference
     allows for that step, its allowed variants included: at most half the smallest value it
     allows, or at least twice the largest.
4. Write each error so that it can only be read as its own type, never as another of the
     error types above.
5. When an error lists more than one type, put each one in through its own change, so that
     each can be recognised on its own, and do not let them contradict each other (a 30-
     second heating and a burnt result).
6. For a correction, rewrite the lines of its step so that the step goes wrong as a slip, and
      write the corrective action as a new action of its own: it only undoes or makes up for
     the slip and does not carry out any other step of the reference. Place it after the step
      that slips or after a later step.
7. Only a correction and an addition add a new action; every other error only changes or
     removes lines of its step. A slip that is not a correction is not corrected: no
     corrective action follows it and the rest of the video goes on as written, so choose a
     slip that the later lines still fit (if the water goes into another bowl, the kettle
     stays empty and a later line pouring from the kettle no longer fits).
8. For an addition, write one extra action on the food or the items of this dish -- adding an
      ingredient the dish does not call for, processing something one more time, or handling
     an item in a way the procedure never asks for -- that is not a step of the reference,
     nor a repeat or a part of one, nor an allowed variant listed in the reference, and place
      it after whichever step it fits.
9. An error changes or removes only lines within its own lines. The one exception: when you
     change an item (an ingredient, a tool, a container, a count), another line -- earlier or
      later, also a line of another error's step -- that names the same item in a way the
```

```
     change makes untrue must be changed to match, replacing only those words; put such a
     line in <carried> (a line appears in one block only). Do not add the change to a line
     that does not name the item that way (after a dirty egg, "whisks the egg" stays as
     written). Keep every other line exactly as written.
10. Give every line you change or add its duration in seconds, judged from the durations
     shown: a changed line keeps its present duration unless the error changes how long it
     takes (a timing error always does); a new line takes as long as it would really take.
11. Write like the captioner: the same tense, person and phrasing as the other lines.

Return only the following, and nothing else. One <error> block for every error, in the order
     of the list, with its id, step and type copied from the list; then one <carried> block (
     empty if nothing is carried). Inside a block:
- <line n="..." seconds="...">the whole new text of that line</line> for a line you change,
- <remove n="..."/> for a line you remove,
- <new_action after="..."> with one <line seconds="...">text</line> per new line, for the new
      action of a correction or an addition; "after" is the step label it follows, written
     exactly as in the captions (A4, or A13 (2)).
For example:
<error id="1" step="A3" type="measurement">
<line n="12" seconds="6">scoops sweet and sour sauce from the bottle with a 1-cup measuring
     cup</line>
<remove n="13"/>
</error>
<error id="2" step="A5" type="correction">
<line n="18" seconds="5">pours the hot water from the kettle into a bowl beside the mug</line
     >
<new_action after="A5">
<line seconds="5">pours the water from the bowl into the mug</line>
</new_action>
</error>
<error id="3" type="addition">
<new_action after="A7">
<line seconds="4">sprinkles salt into the mug</line>
</new_action>
</error>
<carried>
<line n="25" seconds="4">places the salt shaker back on the table</line>
</carried>
```

### I.7 DETECTOR

Qwen3.5-9B, zero-shot and fine-tuned, one call per test video; the same prompt is the input of every SFT example. {types} lists the definitions of the dataset's step-level error types.

**CaptainCook4D.**

```
Below is the procedural reference for {dish}, and the dense captions of a video of a person
     following it. The captions were written by an automatic captioner, in temporal order,
     and an automatic segmenter grouped them into step blocks: each block is numbered, shows
     its time span and the reference step it was assigned, or "background", and lists its
     caption lines with their times. Judge, block by block, whether the person carries out
     the procedure correctly, then whether any step of the reference was left out.

Procedural reference:
{reference}

Step blocks:
{blocks}

The error types of this dataset:
{types}

Rules:
- Judge each block by what its caption lines state, compared with the reference step it
     carries out: the ingredient, tool, container, amount, duration, setting, manner and
     outcome. Only a stated detail that contradicts the reference counts; a detail the
     captions do not mention is not evidence, and anything listed under Allowed Observational
      Variants is not an error.
- Each block carries out the reference step it is labelled with; judge how that step is
     carried out. A background block is normally normal; flag it only when its lines
     themselves show one of the error types.
- A step done with a wrong ingredient, tool, container or amount, or done in the wrong order,
      is done, not omitted. A missing step is never a block's error: it goes in the final
     block only.
- Order: follow the running order of the steps carried out against the Ordering Constraints.
     A block is an order error when it carries out a step before a step that must come first,
      or after one that must come later; the Explanation names the constraint.
- One block may carry several errors of different types; list them all.
```

```
- The Explanation says in one sentence what the person did and what the reference asks for.
- OmittedActions lists a reference step, by its id and text, when no block carries it out at
    all; InsertAt gives, for each, the time (mm:ss) at which it should have begun: the end
    of the last completed step that must come before it in the reference order (its nearest
    performed predecessor), or 00:00 when none of those steps was done; the final block's
    Explanation says in one sentence per omitted step what the reference asks for, why the
    captions show it was never done, and where it belongs: "right after Step k (A<id>), <
    what that block does>", or "at the start" when none of its preceding steps was done.

Answer with exactly one block per step block, in order and with the same numbers, then one
    final block, and nothing else. A correct step gets only its verdict:
<step1>
Verdict: normal
</step1>
<step2>
Verdict: error
ErrorType: <one of the types above; several, separated by commas, when the block carries
    several>
Explanation: <one sentence>
</step2>
...
<final>
OmittedActions: <none, or the omitted steps as "A<id> <step text>", separated by semicolons>
InsertAt: <none, or one time mm:ss per omitted step, in the same order, separated by
    semicolons>
Explanation: <none, or one sentence per omitted step, in the same order, separated by
    semicolons: what the step requires, that no block shows it, and right after which step
    block it belongs>
</final>
```

**EgoPER.**

```
Below is the procedural reference for {dish}, and the dense captions of a video of a person
    following it. The captions were written by an automatic captioner, in temporal order,
    and an automatic segmenter grouped them into step blocks: each block is numbered, shows
    its time span and the reference step it was assigned, or "background", and lists its
    caption lines with their times. Judge, block by block, whether the person carries out
    the procedure correctly, then whether any step of the reference was left out.

Procedural reference:
{reference}

Step blocks:
{blocks}

The error types of this dataset:
{types}

Rules:
- Judge each block by what its caption lines state, compared with the reference step it
    carries out: the ingredient, tool, container, amount, duration, setting, manner and
    outcome. Only a stated detail that contradicts the reference counts; a detail the
    captions do not mention is not evidence, and anything listed under Allowed Observational
     Variants is not an error.
- Each block carries out the reference step it is labelled with; judge how that step is
    carried out. A background block is normally normal; flag it only when its lines
    themselves show one of the error types.
- A step done with a wrong ingredient, tool, container or amount is done, not omitted. A
    missing step is never a block's error: it goes in the final block only.
- A step done out of order is not an error of this dataset: judge each block by how it is
    carried out, not by its position.
- When a step goes wrong and a later action makes up for it, the block that goes wrong is a
    slip and the block that makes up for it is a correction. An addition is a block whose
    action is not a step of the reference, nor a repeat or part of one, nor housekeeping
    around a step.
- Each block carries at most one error type.
- The Explanation says in one sentence what the person did and what the reference asks for.
- OmittedActions lists a reference step, by its id and text, when no block carries it out at
    all; InsertAt gives, for each, the time (mm:ss) at which it should have begun: the end
    of the last completed step that must come before it in the reference order (its nearest
    performed predecessor), or 00:00 when none of those steps was done; the final block's
    Explanation says in one sentence per omitted step what the reference asks for, why the
    captions show it was never done, and where it belongs: "right after Step k (A<id>), <
    what that block does>", or "at the start" when none of its preceding steps was done.

Answer with exactly one block per step block, in order and with the same numbers, then one
    final block, and nothing else. A correct step gets only its verdict:
<step1>
Verdict: normal
</step1>
```

```
<step2>
Verdict: error
ErrorType: <one of the types above>
Explanation: <one sentence>
</step2>
...
<final>
OmittedActions: <none, or the omitted steps as "A<id> <step text>", separated by semicolons>
InsertAt: <none, or one time mm:ss per omitted step, in the same order, separated by
     semicolons>
Explanation: <none, or one sentence per omitted step, in the same order, separated by
     semicolons: what the step requires, that no block shows it, and right after which step
     block it belongs>
</final>
```

### I.8 TEACHER

GPT-5.6 Terra. The teacher prompt is the detector prompt of the variant, with its sentence "Answer with exactly one block per step block, in order and with the same numbers, then one final block, and nothing else." replaced by "After your reasoning (see below), answer with exactly one block per step block, in order and with the same numbers, then one final block.", followed by the text below. {answer_key} is the answer key described in Appendix C.1.

```
The correct answer for this video is given below. Your task is to write the reasoning that a
     careful observer would go through to reach it from the reference and the captions, and
     then the answer itself.

<answer_key>
{answer_key}
</answer_key>

Begin your reply with the reasoning, written between <think> and </think>. Go through the
     step blocks in order, starting the part on each with "Step N:".
- A block the answer key marks normal: say which reference step it carries out and why what
     its captions state fits that step or one of its Allowed Observational Variants.
- A block the answer key marks as an error: write (1) what the reference step requires that
     matters here -- the ingredient, tool, container, amount, duration, setting, order or
     result; (2) the caption words that show the deviation, quoted exactly; (3) for an order
     error, which ordering constraint is broken and by which other step block; for a
     correction, which earlier step block went wrong, how, and how this block makes up for it;
      for an addition, why the action is not part of any reference step.
- Only when the match is not clear -- the block's label does not fit what its captions show,
     the captions are vague, a detail is close to an allowed variant, or the block mixes two
     steps -- say what makes it unclear, weigh the possible readings, and explain which one
     the captions support. Where the match is clear, do not invent doubt.
Then check which reference steps no block carries out, arriving at the omitted steps of the
     answer key and at where each one belongs.
Write the reasoning as someone finding this out from the captions for the first time: never
     mention the answer key, that an answer was given, or that the video or its captions were
      edited, injected or synthesized. After </think>, give the answer in the format above,
     with exactly the verdicts, error types, omitted steps and InsertAt times of the answer
     key and an Explanation of your own for each error and for each omitted step -- the
     omitted step's sentence must end by placing it where the answer key puts it, as "it
     belongs right after Step k (A<id>), <what that block does>" or "it belongs at the start"
      -- and nothing after it.
```

### I.9 PAIRWISE EXPLANATION JUDGE

GPT-5.6 Terra, called on each ground-truth error segment that both systems detect, once for each order of the two explanations (Sec. 4.1). {question} is filled with the question of the segment's error type below, whose {x} and {y} are the annotated step and error description; lines starting with # are comments of the file.

**Judge prompt.**

```
Below are two explanations, A and B, of one mistake in a video of a person preparing {dish}.
     Each was written by an automatic system that watched the video. You cannot see the video.
      The mistake itself is given in the question; it comes from the dataset's human
     annotation and is correct.

Steps of this task (the dataset's official step list{order_note}):
{steps}
```

```
Question:
{question}
Pointing out the mistake matters more than recognizing the step.

Explanation A:
{candidate_a}

Explanation B:
{candidate_b}

Rules:
- An explanation points out the mistake if it describes the same event as the question, in
     any wording. Different words, more or less detail, and writing style (for example "you"
     or "the person") do not matter.
- An explanation that describes a different event, only names a category of mistake, or only
     says that something went wrong, has not pointed out this mistake.
- Do not prefer an explanation for being longer or more fluent. A and B are in an arbitrary
     order.
- You must choose A or B. No tie is allowed.

Return exactly two lines and nothing else:
Winner: <A or B>
Reason: <one sentence>
```

**Questions per error type.**

```
# One question per error type, filled into {question} of pairwise.txt.
# <x> = the annotated step name, <y> = the annotated error_description.
# EgoPER uses the type names as they are. CaptainCook4D maps its deciding_tag:
# Measurement / Timing / Temperature / Preparation Error -> modification
# Technique Error -> slip, Order Error -> order, Missing Step -> not scored
slip: Which explanation better recognizes that the person is doing "{x}", and points out the
     mistake "{y}"?
modification: Which explanation better recognizes that the person should be doing "{x}", and
     points out that they did "{y}" instead?
addition: Which explanation better points out that the person did "{y}", an extra action that
      is not part of the task?
correction: Which explanation better recognizes that the person is doing "{x}", and points
     out that they did "{y}" to fix an earlier mistake?
order: Which explanation better recognizes that the person is doing "{x}", and points out
     that it was done in the wrong order: "{y}"?
```