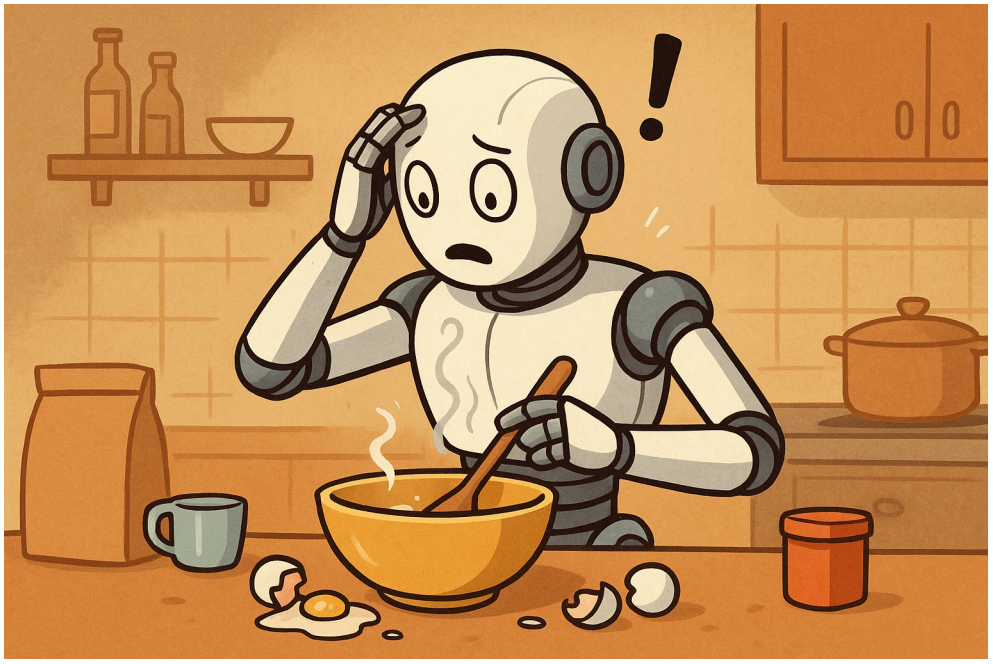
Learning to spot mistakes in procedural activities

Teaching Assistants: Simone Alberto Peirone, Gaetano Salvatore Falco

Link to this document (check for updates) - Current version: 1.0 (5 Nov. 2025)

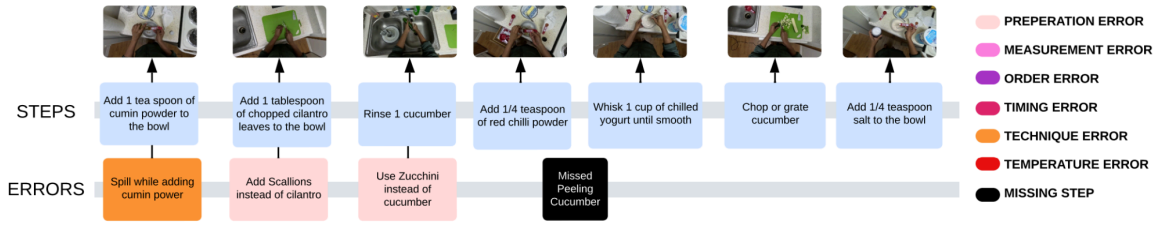
🚨 IMPORTANT:
For assistance during the development of this project, join the specific discord channel.
We will organize weekly meetings until the end of the course to answer your questions.



An illustration of a humanoid robot preparing a recipe and suddenly realizing it made a mistake in the preparation (ChatGPT)

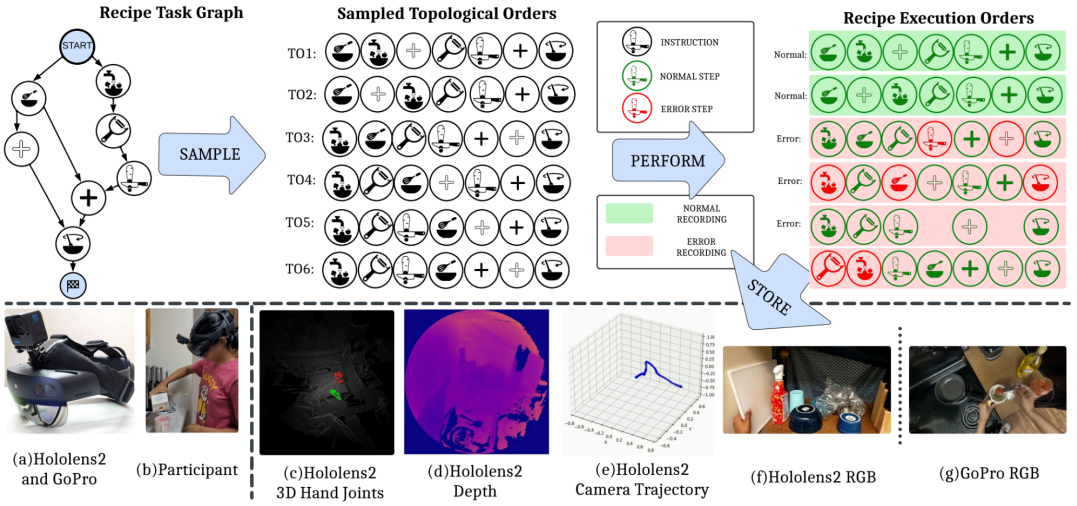
Project overview

Think about your typical morning routine. You wake up and decide to prepare an omelette. Even a simple recipe requires you to follow a sequence of steps (a so-called *procedure*): breaking the eggs, mixing them with the ingredients, and finally cooking the omelette. Some steps can be omitted, maybe you don't want cheese in your omelette, while others can be reordered or interleaved, such as changing the order in which you mix the ingredients. But then you get a call from a friend and forget about the pan. When you come back, the omelette is visibly burnt. Something went wrong in the process, and you need to start over.



A recipe execution with different kind of intentional and unintentional errors

In this project we will focus on *procedure understanding*, a field that deals with teaching models to recognize, segment, and reason about the sequence of actions that compose complex human activities. Most specifically, we will explore the mistake detection task, which involves detecting and recognizing different kinds of errors in the steps of a procedural activity. In this setting, you will learn about *task graphs*, a compact representation of the sequential steps required to complete a recipe. Task graphs are a special form of directed acyclic graphs. They can be manually annotated, or directly learnt from multiple human demonstrations of the same recipe [PL2]. Given a video of a recipe and its corresponding task graph, we can predict whether the recipe was executed correctly or not by verifying that the steps satisfy one of the possible paths of the task graph.



Task graphs and different recipe executions from CaptainCook4D

The project is articulated as follows. In the first part of the project, you will train simple baseline models to predict the correctness of a procedure step from its corresponding video snippet. Then, you will have the chance to develop an extension. In the suggested extension, we move beyond mistake detection to a task verification setting, where the goal is to classify the correctness of an entire video recipe given its corresponding task graph.

Alternatively, you may propose a different extension. Please discuss this with the teaching assistant before you start working on the extension to ensure that everything is clear.

Project setup and technical details

This project is meant to be developed using the free GPU resources provided by Google Colab, an hosted Jupyter notebook platform. Google Colab provides an easy way to connect the notebooks to Google Drive for data storage. Keep your code on Github (forking the project repository) and all the features, videos and annotations on Google Drive. Each time you start up a new notebook, you can copy the data you need from Drive to the local instance your notebook is running on.

Step 1: Literature Review

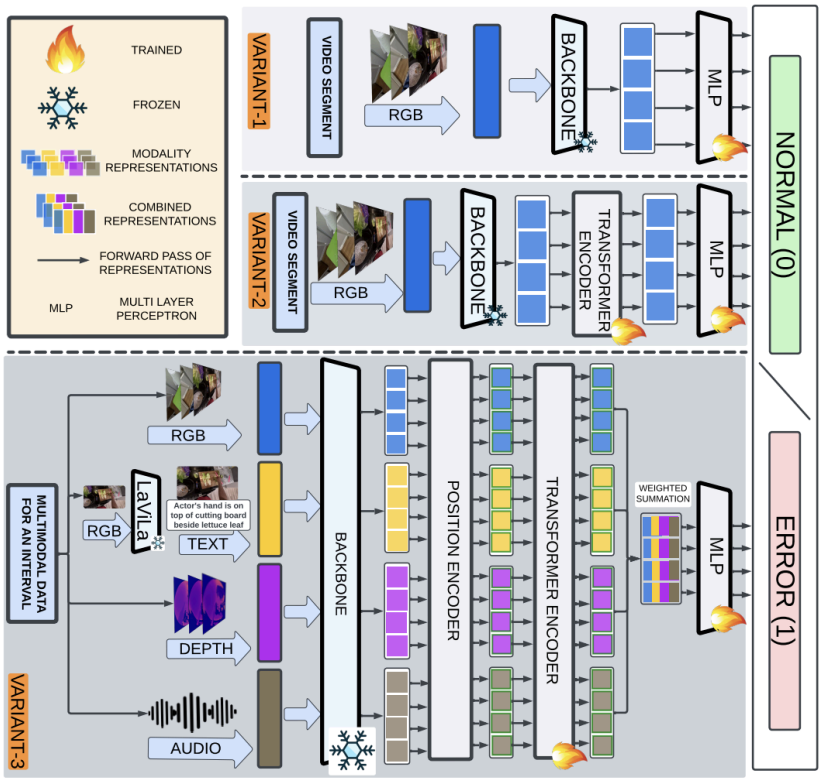
In this step, you should become familiar with the most relevant works in procedural activity understanding. Start from the suggested papers in the Reference section of this document,

focusing on the CaptainCook4D dataset [PL1], and some of the most recent works in Procedure Learning [PL2, PL3]. You don't need to fully understand each paper at this stage, you should just grasp the main concepts.

Step 2: Mistake Detection baselines

TL;DR: In this step, you will train a few simple baseline for offline mistake detection, aiming to reproduce the results reported in the CaptainCook4D paper.

In the Supervised Error Recognition task (SupervisedER), the goal is to classify a video segment (few seconds/minutes) corresponding to a procedure step as either correct or incorrect execution (binary classification task). The CaptainCook4D paper proposes three simple baselines for this task. The first (V_1) is a simple Multi-Layer Perceptron (MLP) head trained on top of pre-extracted sub-segment features. V_2 leverages a Transformer layer to combine the cues from all sub-segments in the recipe step to predict whether the execution is correct or not. V_3 combines visual (RGB) inputs with textual narrations, audio recordings and depth information to further improve the prediction accuracy. For this project, we will consider only baselines V_1 and V_2.



Supervised Error Recognition baselines from CaptainCook4D

Substeps:

1. 📌 Download the pre-extracted features. Training video models is very computationally expensive. As a result, we typically overcome this issue by leveraging transfer learning, reusing models trained on large scale datasets with some generic training objectives (e.g., action recognition or contrastive video-language alignment) to extract high-level representations of the input video.

Following the CaptainCook4D paper, we consider the following pre-trained backbones: Omnivore and SlowFast. Pre-extracted features are provided as part of the dataset release and you can download them from here.

   a. Look for details in the CaptainCook4D paper describing the features extraction process. What are the input and outputs of the pre-trained models in this phase?

2. 📌 Reproduce the the V_1 and V_2 baselines from the CaptainCook4D. In this step you should replicate the results of the V_1 (MLP) and V_2 (Transformer-based) baselines on the SupervisedER task from the original paper, using the same metrics (Accuracy, Precision, Recall, F1 and AUC).
   a. In addition to the metrics from the original paper, you should analyze the performance of the model on different error types.
   b. Propose a new baseline for this task and compare with the existing ones. For example, you could train an RNN/LSTM on the sequence corresponding to each step.

3. 📌 Extend the baselines to a new features extraction backbone. Adapt the features extraction code provided by CaptainCook4D to support a new features extraction backbone. Some of the backbones you may consider include EgoVLP [VB4] and PerceptionEncoder [VB5]. You can download a resized version of the dataset from here.

Extension: From Mistake Detection to Task Verification

TL;DR: in this extension we move beyond mistake detection to a task verification, where the objective is to predict whether an entire video recipe represents a correct execution.

🚨Please discuss the extension with the TA before starting!

Detecting mistakes could be useful, but ultimately we care about whether the recipe was correctly completed as a whole. In this extension, we move beyond mistake detection to a broader task verification setting, where the objective is to predict whether a given recipe video corresponds to a correct or incorrect execution, by looking jointly at the video and the corresponding task graph. Unlike mistake detection, this formulation does not require step-level annotations about the correctness of the execution of the individual steps. Instead, we only need a single recipe-level binary label indicating whether the overall execution is correct or incorrect.

In this extension, we consider a three stage pipeline. First, we use a pre-trained step localization network to detect individual steps in an input recipe video. Then, we match each detected step to the corresponding node in the task graph. *Task graphs represent all possible (correct) executions of a recipe. By matching the task graphs with the detected steps in the video, we obtain a "realization" of the task graph.* Finally, we learn a classifier to predict whether the observed task graph corresponds to a correct or incorrect recipe.

⚠️ For this extension, you should use the EgoVLP or PerceptionEncoder features you extracted in the Step 2 of the project. For these encoders, the video and text embedding

spaces are aligned, allowing us to match the visual features of the steps to the textual encoding of the steps in the task graph.

Substep 1: Recipe step localization

In this first step, you should use a step localization approach to segment the individual steps of a recipe video. You can use different approaches, like the pre-trained ActionFormer model provided as part of CaptainCook4D, or a zero-shot clustering-based approach like HiERO [PL4]. The output of the model is a list of tuples *(start, end)* that identify the start and end timestamps of each step in the video.

Then, for each step you should compute a unique step-level embedding by averaging all the video features that fall within the (start, end) boundaries of the step. As a result, for each video you will have a sequence of step-level embeddings, summarizing the visual features of the detected steps.

Substep 2: Simple Task-Verification baselines

In this step, you should train a simple baseline for task-verification on the recipe videos using binary labels to identify correct and incorrect executions. For example, you can define the model as a transformer layer that takes the sequence of step-level embeddings, followed by a binary classification head (you can take inspiration from the Transformer baseline for error recognition).

Given the limited number of samples in the dataset, you can consider a leave-one-out evaluation setting in which the model is trained on (k-1) recipes and tested on the k-th recipe.

Substep 3: Task-Graph encoding + Step matching

In this step, you should take the ground truth task graphs provided by CaptainCook4D and encode the textual descriptions of their steps using the EgoVLP/PE textual encoder. The features of the task graph's nodes and the visual features extracted in Substep 1 are aligned, hence we can find for each step in the video the best matching node in the task graph and vice versa. You can assume each visual step can be matched to at most one node (and vice versa) and use the Hungarian matching algorithm to match all the steps to the nodes.

Then, update the features of the matched nodes with a learnable projection of the node features and the visual features.

Substep 4: Classification of the observed task-graph

In this final step, you should train a simple Graph Neural Network (GNN) based classifier to predict whether the execution is correct or not from the realization of the task-graph. Use this notebook as a reference for the training. You are free to experiment with different GNN layers. For example, DAGNN [O1] is a layer specifically designed for directed acyclic graphs.

Project Deliverables

For the exam, you are expected to produce a report of your project using the CVPR template. The report must be 8 pages-long and follow the typical paper-like structure

(abstract, introduction, related works, method, experiment and conclusion). Clearly explain everything you did in the project, focusing on the implementation of the extension.

References

Procedure Learning (PL):

1. Peddi, Rohith, et al. "CaptainCook4D: A dataset for understanding errors in procedural activities." NeurIPS 2024.
2. Seminara, Luigi, et al. "Differentiable Task Graph Learning: Procedural Activity Representation and Online Mistake Detection from Egocentric Videos." NeurIPS 2024.
3. Flaborea, Alessandro, et al. "PREGO: online mistake detection in PRocedural EGOcentric videos." CVPR 2024.
4. Peirone, Simone Alberto, et al. "HiERO: understanding the hierarchy of human behavior enhances reasoning on egocentric videos." ICCV 2025.
5. Bansal, Siddhant et al. "My view is the best view: Procedure learning from egocentric videos." ECCV 2022.
6. Chowdhury, Sayeed Shafayet et al. "Opel: Optimal transport guided procedure learning." NeurIPS 2024.

Video Backbones (VB):

1. Girdhar, Rohit, et al. "Omnivore: A single model for many visual modalities." CVPR 2022.
2. Feichtenhofer, Christoph, et al. "Slowfast networks for video recognition." ICCV 2019.
3. Girdhar, Rohit, et al. "Imagebind: One embedding space to bind them all." CVPR 2023.
4. Lin, Kevin Qinghong, et al. "Egocentric video-language pretraining." NeurIPS 2022.
5. Bolya, Daniel, et al. "Perception encoder: The best visual embeddings are not at the output of the network." arXiv preprint arXiv:2504.13181 (2025).

Others (O):

1. Thost, Veronika et al. "Directed Acyclic Graph Neural Networks." International Conference on Learning Representations.