How to Correctly Make Mistakes: A Framework for Constructing and Benchmarking Mistake Aware Egocentric Procedural Videos

Supplementary Material

A. Egocentric Procedural Video Datasets (Context)

In this section, we review the main procedural video datasets, both with and without errors. Beyond a high-level comparison (Table 1), we describe in more detail the datasets that are central to this work: **EgoPER, EgoOops, Assembly101, CaptainCook4D, and Ego-Exo4D keysteps**.

Ego-Exo4D keysteps is used as source procedures for PIE-V. EgoPER, EgoOops, Assembly101, and CaptainCook4D serve as real-data references for what mistakes and corrections look like under different annotation schemes.

Most procedural datasets were created primarily for action recognition, action segmentation, key step (sequence) extraction, object interaction, or pose estimation based on visual data. Consequently, instead of a full description of the steps, the annotations may take the form of action labels. This is typical of early datasets in the assembly domain such as MECCANO [20], Assembly101 [23], ATA [6], and IndustREAL [22]. A shorter description of the procedure step makes it harder to recognize errors based on semantic cues.

The error annotations are heterogeneous and fragmented. The first group of general error annotations treats mistakes purely at the level of sequence validity, i.e., whether the overall execution follows the canonical procedure, without localizing or typologizing individual erroneous steps. In CSV [19] each video of an experiment is labeled as correct or incorrect with respect to the entire reference protocol. ATA [6] focuses on detecting whether an activity sequence adheres to the expected order, emphasizing structural deviations such as deletions. The second group extends step/action annotations with per-step binary correctness labels. In HoloAssist [25], action segments are labeled as "correct" or "mistake", and conversational interventions are categorized (e.g., corrections, follow-ups), but the error label itself does not distinguish between procedural and executional issues. Assembly101 [23]-based benchmarks used in Ding et al. [5] attach a binary mistake flag to specific action segments and further distinguish structural errors such as misordering or redundant steps, along with incorrect attachment of parts. Notably, this benchmark also marks accumulating mistakes and corrective steps (detaching incorrectly attached parts) with a special label.

A smaller number of datasets introduce explicit taxonomies of both structural and execution errors, often tied to a specific domain. EgoPER [12] defines five error types assigned at the step level (omission, addition, modification, slip, correction). CaptainCook4D [18] provides a cooking-specific taxonomy (measurement, timing, temperature, technique, missing and misordered steps). EgoOops [8] adopts another multiclass execution-error taxonomy (working with wrong objects, grasping wrong objects, correction, unintended actions, working in the wrong way, and others). CaptainCook4D and EgoOops augment each erroneous segment with a natural-language explanation of the error aligned to the procedural text (EgoOops additionally sometimes captures correction behavior in free-text explanations).

Assembly101. In the Assembly101 annotations, each step is represented by one action class and two object classes that are being manipulated. There are only two actions: *attach* and *detach*[1]. The full object vocabulary contains 64 parts, and some of them are semantically close (for example, "roller arm", "crane arm", and "excavator arm" can all be seen as instances of a more general "arm").

The classes only record the action and the objects, so if an object is attached incorrectly (with a wrong orientation), this can be seen only from the error-type label "wrong orientation". To make the distinction clear at the text level, we converted class labels into full imperative commands using templates, for example, "attach the step to the chassis in the wrong way" versus the correct "attach the step to the chassis". In the original annotations, there is also no consistency between attaching part X to part Y and attaching part Y to part X. We normalized such steps to a single canonical form. As a result, we obtain a vocabulary of 339 full step descriptions.

Some toy variants may have only a single assembly in the whole dataset (e.g., `c13c` for a single correct assembly and `b04b` for a single erroneous assembly). Since the toy subtypes encoded by the last letter in the `toy_id` differ only slightly, we merge them into shared type classes.

Overall, the error annotations in Assembly101 do not fully match our taxonomy, because the original annotations follow the logic of the assembly process rather than the logic of conformity to a reference procedure. For example, from the assembly point of view, detaching an incorrectly attached part can be a correct step, but from the point of view of matching the canonical assembly, no detachment

[1] In the original annotations there is a third rare verb class, "position", which appears only together with the object "figurine". For the sake of simplicity it was merged into "attach".

Table 1. Egocentric procedural video datasets. **#Steps** refers to the number of distinct action/step classes when reported; otherwise it is left as "–". For **Step annotations**, *step* refers to natural step descriptions, while *action label* refers to verb + object(s) phrases. *Timestamps* mark either time or frame stamps.

Dataset	#Videos	Duration [h]	#Tasks	#Steps	Domains	Step annotations	Mistakes	Source
MECCANO [20]	32	–	1	–	toy assembly	action labels + timestamps	×	controlled lab
EPIC-KITCHENS-100 [4]	∼700	100	–	> 90k	cooking, kitchen activities	action labels + timestamps	×	participant recordings
50 Salads [15]	50	∼6.4	1	17	cooking	action labels	×	controlled lab
EgoProceL [1]	329	62	16	139	various, incl. cooking, assembly	steps + timestamps	×	semi-controlled participant recordings
HoloAssist [25]	350	166	20	414	AR-assisted manipulations, incl. assembly	summary, conversations, steps + timestamps	✓	controlled lab
Assembly101 [23]	362	167	101	202	toy assembly	action labels + timestamps	✓	controlled lab
CaptainCook4D [18]	384	94.5	24	352	cooking	steps + timestamps	✓	participant recordings
EgoOops [8]	50	6.8	5	46	lab-style experiments and controlled assembly tasks	step + timestamp	✓	controlled lab
Ego-Exo4D (keysteps) [7]	852	30	17	186	various, incl. cooking, repair	step + timestamp	✓†	participant recordings
EgoPER [12]	396	28	5	70	cooking	step + timestamp	✓	participant recordings
ATA [6]	141	24.8	3	15	toy assembly	action labels	✓	controlled lab
EPIC-Tent [9]	24	> 5.4	1	38	assembly	action labels + timestamps	✓	participant recordings
CSV [19]	70	11.1	14	106	chemical experiments	action labels	✓	controlled lab
IndustReal [22]	84	5.8	2	75	toy assembly	steps + timestamps	✓	Industrial-like lab

† † Only 17 keysteps carry the `Mistake` label; it replaces the step description, so we use Ego-Exo4D keysteps as a clean source and inject mistakes synthetically.

should be considered a correct step. Similarly, re-attaching a part after an erroneous detachment may be labeled as a *Correction*, but with respect to the reference procedure this step is simply correct.

From the visual point of view, actions in Assembly101 are mirror-like: a detachment is the reverse of an attachment of the same parts. The parts in the dataset are also visually specific: they are sometimes small and visually similar to each other. In addition, the visual referent of the same object can change as the assembly progresses. For example, in assembly `c03f`, at the step "attach the arm connector to the chassis" the chassis has one appearance, while in the next step, "attach the body to the chassis", the term "chassis" refers to two already connected parts, the arm connector and the chassis.

CaptainCook4D. A key property of the cooking domain is the need to follow precise quantities to execute a recipe successfully. However, for visual models it is hard to see the difference between, say, "Add 1/3 tsp salt to the pan" (step_id: 178) and "Add 1/2 tsp salt to the pan" (step_id: 146), and even for a human this is often unclear from a single video. The dataset authors also note in Peddi et al. [18] that full recipe understanding is multimodal rather than purely visual.

Even so, some dataset steps have very similar textual descriptions but different step IDs. For example, "Take 1 tomato" (step_id: 149) versus "Take a tomato" (step_id: 247), or "Peel 1 garlic cloves" (step_id: 200) versus "Peel 1 garlic clove" (step_id: 14). Such steps are visually indistinguishable and identical in their semantic representations. For each erroneous step, the modified textual description also provides corrected descriptions. However, this is not consistent. For example, in recording `1_33`, the first step "Coat a 6-oz. ramekin cup with cooking spray" is labeled with the preparation error "Coating a large bowl instead of 6-oz ramekin cup", yet later steps are marked as correct and described as "Microwave the ramekin cup uncovered on high for 30 seconds", "Stir the ramekin cup" and so on. In the video the same bowl appears in all steps. Such inconsistencies in the step descriptions create discrepancies between the modalities, which are reflected in our Text–Video Grounding Consistency metric.

Some steps also share almost the same temporal segment, but receive different textual descriptions and different step IDs. For example, in `2_28` the step "Cut 1/8 garlic clove" is annotated from 564.6 to 624.6 and the step "Mince 1/8 garlic clove" – from 582 to 640. These steps are easy to distinguish in text but almost indistinguishable visually, adding further misalignment between the visual and textual modalities.

Given that each dataset contains less than 400 videos,

these numerous inconsistencies contribute a substantial amount of noise for the models. This motivated us to include it in the list of datasets for annotators' assessment.

EgoPER. EgoPER [12] is an egocentric cooking dataset built around a small set of recurring recipes (coffee, quesadilla, pinwheels, tea, oatmeal). We stratified 5 videos of each task for our annotators' assessment. The given dataset annotations include step-level timestamps and one of five labels of the following taxonomy: two structural deviations (step *omission* and step *addition*), two execution-level deviations (*slip* and *modification*), and *correction*. A distinctive aspect of EgoPER is that the annotations separate task-relevant steps from background activity: some segments reflect incidental actions that are not part of the core procedure (e.g., reading a script on a screen), and are marked explicitly as background rather than being forced into the step taxonomy. This design is helpful for studying mistake detection without conflating procedural steps with incidental context.

EgoOops. EgoOops [8] contains 5 tasks with 10 videos per task. While it was designed to include both mistake-free executions and scripted mistake executions, in practice additional small deviations also appear in the "correct" runs (e.g., extra grasping or redundant manipulation), which makes the boundary between benign variation and mistake-like behavior particularly salient (reflected in our Error Validity metric). EgoOops gives a multi-class taxonomy of deviations (including corrections) and aligns each execution to a canonical script. However, for error segments the text often describes the *deviation from the canonical step* rather than the exact action the person performed. For example, instead of restating the full step description, the annotation may specify what was wrong relative to the canonical step with a description such as "correct errors in steps 1 and 2". This complicates purely text-based assessment: recovering the implied correct step may require broader context and/or video grounding. Additionally, some steps are semantically dense and contain multiple predicates (e.g., "pour ... then dip ... and squeeze ..."), which increases the structural complexity of the instruction.

Ego-Exo4D keysteps. Ego-Exo4D [7] is a large-scale egocentric/exocentric dataset with multiple benchmarks. For PIE-V, we specifically take the split of the keystep annotations because they provide (i) step timestamps and (ii) natural-language step descriptions suitable as inputs for controlled textual rewriting.

A practical property of the keystep annotations is that step structure can be hierarchical: a step may be a leaf or a parent over finer-grained children, and some steps are explicitly marked as non-essential (i.e., present in the video but not required for the canonical goal). In PIE-V we use all available steps when constructing clean source procedures, because this reduces the risk of deleting or transposing critical actions when injecting errors, and it preserves a faithful "what happened" trace. We leave for future work a more aggressive setting that injects mistakes only into essential keysteps.

B. Details on Annotators and Annotations

Annotator pool and diversity of judgments. We use five annotators (2 male, 3 female; age 20–47) with heterogeneous backgrounds (engineering and humanities) and educational levels ranging from high school to graduate training (including two Master's degrees and one PhD). This diversity was intentional: most rubric dimensions are designed to capture *human perception* of mistake realism and coherence rather than a single objective ground truth. In particular, Human Plausibility, Confusability, and Video Plausibility are subjective judgments by design, while Error Validity and Taxonomy Fit are expected to be more stable across annotators.

Annotation workflow and interface. Annotators evaluated samples in a paired setting with a reference (mistake-free) execution and a mistake-aware execution shown side by side. Mistake and correction steps were explicitly highlighted in the evaluated trace; the task was to *rate the indicated deviations and recoveries*, not to discover them. The annotation interface (implemented in structured Google Sheets templates) provided metric-specific inline guidance and drop-down fields for each rating (Fig. 1). Reference and erroneous procedures were aligned stepwise to support direct comparison.

Metric design philosophy. With the exception of Taxonomy Fit (error category assignment), the rubric metrics were designed to capture different aspects of perceived error naturalness and procedural coherence. Error Validity serves as a gate-like metric that distinguishes consequential procedural mistakes from non-consequential deviations; disagreement on this binary judgment captures boundary cases. The remaining metrics decompose human judgments into plausibility (text/video), noticeability (Confusability), sequence-level coherence, and world-state consistency, rather than collapsing them into a single score.

Annotator instructions and scale interpretation. Each metric was presented with a short operational definition and anchored response options. For example, Human Plausibility was defined as whether the described deviation looks like a mistake a real person could make in the given context, while Confusability measured how easy it would be

Table 2. Audit sampling summary for the four existing mistake-aware datasets (25 videos each).

Dataset	Videos	Total steps	Mistake steps	Mistake rate	Avg. steps/video	Avg. mistakes/video
EgoPER	25	358	134	37.43%	14.32	5.36
EgoOops	25	302	87	28.81%	12.08	3.48
Assembly101	25	409	166	40.59%	16.36	6.64
CaptainCook4D	25	370	192	51.89%	14.80	7.68

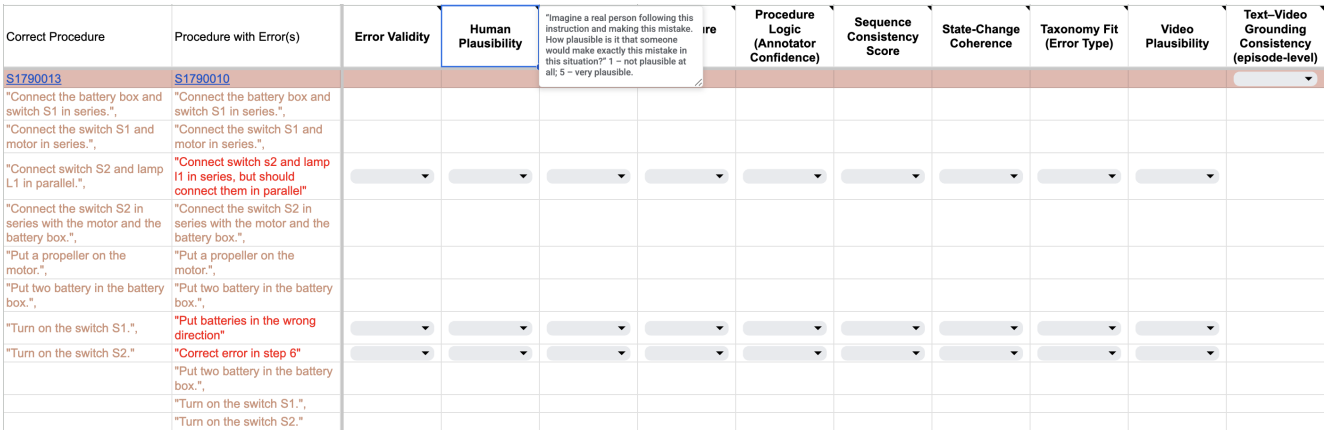

Figure 1. Annotation interface used for the rubric (example from the electronics task in EgoOops). Annotators are shown a **reference** mistake-free execution (with video) alongside a **mistake-aware** execution containing **mistakes and corrections** (highlighted in red). The goal is not to localize errors but to **rate the indicated deviations** on step-level and procedure-level criteria. Each metric includes a short in-UI explanation, and ratings are selected via drop-down menus.

to overlook the mistake during real-time task execution. Procedure Logic was collected as a binary judgment (does the mistake make the textual procedure logically inconsistent as a whole) together with a 3-level confidence rating, which is later combined into a confidence-weighted aggregate score. Sequence Consistency and Text–Video Grounding were rated on 5-point Likert scales. State-Change Coherence was annotated as a binary judgment on whether the text implies any clear inconsistency in object identity, availability, or state transitions.

Pilot calibration and agreement monitoring. We did not expect uniformly high agreement across all metrics, because several dimensions are intentionally subjective. Instead, we used Krippendorff's α as a monitoring signal and iteratively refined annotation guidelines after a pilot round, especially for metrics that produced strong outliers or systematic disagreements. This calibration focused on clarifying metric wording and decision boundaries (e.g., procedural error vs. harmless variation; Substitution vs. Wrong Execution), not on forcing consensus.

Agreement summary. Inter-annotator agreement for all datasets and generation settings is reported in Table 3. Agreement is generally highest for Error Validity and Taxonomy Fit, indicating that annotators largely agree on what counts as a procedural error and how to assign taxonomy labels. Lower agreement on Human Plausibility and Confusability reflects genuine variation in human judgments about realism and noticeability, which is an intended property of these metrics rather than a failure of the annotation protocol.

C. Details on the Method

C.1. Phase Modeling and Phase-Conditioned Priors

PIE-V uses a three-phase abstraction to model *when* errors are likely to occur and *which* error types are more likely in different parts of a procedure. The phase design follows cognitive error accounts that relate error profiles to planning demands, attentional load, routine execution, and post-completion vulnerability [2, 3, 10, 16, 17, 21].

We model three coarse phases: **Phase 1 (initiation)**, where plan construction and low familiarity increase attention demand; **Phase 2 (routine execution)**, where automaticity rises but cognitive load and monotony can produce slips and sequencing failures; and **Phase 3 (completion)**, where attention often drops and omissions become more likely due to post-completion effects.

Table 3. Krippendorff's α agreement summary for all settings. "–" indicates values are unavailable (due to nonexistent videos).

Dataset	Err.Valid	Human Pl.	Confus.	Proc.Logic	Seq.Cons.	State-Chg	Taxonomy Fit	Vid.Pl.	T–V Gr.
EgoPER	0.912	0.541	0.368	0.728	0.628	0.579	0.759	0.574	0.662
EgoOops	0.916	0.592	0.375	0.836	0.667	0.600	0.882	0.579	0.560
Assembly101	0.859	0.584	0.697	0.739	0.649	1.000	0.931	0.670	0.861
CaptainCook4D	0.694	0.758	0.847	0.621	0.542	0.584	0.791	0.550	0.488
Ego-Exo4D-Qwen (freeform)	0.568	0.539	0.417	0.579	0.543	0.643	0.820	–	–
Ego-Exo4D-GPT-5.2 (freeform)	0.701	0.424	0.341	0.593	0.652	0.547	0.752	–	–
Ego-Exo4D-Qwen-PJ (PIE-V+Qwen2.5, Qwen3-VL-judged)	0.958	0.483	0.358	0.631	0.706	0.601	0.905	–	–
Ego-Exo4D-GPT-5.2-PJ (PIE-V+GPT, judged)	0.913	0.489	0.387	0.672	0.619	0.696	0.803	0.630	0.930

Load-based phase boundaries. PIE-V computes step load with Eq. 2 of the main paper, forms the cumulative load over steps, and assigns PHASE_1, PHASE_2, and PHASE_3 by splitting the cumulative load into three equal parts.

Phase error-rate model (where errors occur). PIE-V separates phase-level risk from error type choice. The phase error-rate model specifies relative rates

$$r_{\text{PHASE\_1}} = 0.10, \quad r_{\text{PHASE\_2}} = 0.19, \quad r_{\text{PHASE\_3}} = 0.14,$$

which are then normalized to multipliers with mean 1 before step-index sampling (Eq. 3 of the main paper). In the current implementation, this corresponds approximately to

$$m_{\text{PHASE\_1}} = 0.698, m_{\text{PHASE\_2}} = 1.326, m_{\text{PHASE\_3}} = 0.977.$$

Thus, the middle phase is sampled more aggressively for error placement, while total error count remains controlled separately by procedure-level risk and hard caps.

Phase-conditioned type priors (which errors occur). Conditioned on a selected step, PIE-V samples the error type from phase-specific priors over $\{\text{WE}, \text{D}, \text{S}, \text{I}, \text{T}\}$. The implementation uses unnormalized weights (in this order):

$$\text{PHASE\_1} : [3.5,\ 1.0,\ 2.5,\ 2.0,\ 1.0],$$

$$\text{PHASE\_2} : [2.0,\ 2.0,\ 1.5,\ 2.5,\ 2.0],$$

$$\text{PHASE\_3} : [3.5,\ 2.5,\ 1.0,\ 2.0,\ 1.0].$$

After normalization, the corresponding type probabilities are as shown in Table 4.

Table 4. Normalized phase-conditioned type priors used in the current PIE-V implementation (before feasibility modifiers).

Phase	WE	D	S	I	T
Phase 1	0.35	0.10	0.25	0.20	0.10
Phase 2	0.20	0.20	0.15	0.25	0.20
Phase 3	0.35	0.25	0.10	0.20	0.10

These priors are then modulated by feasibility constraints and step properties (e.g., procedure length, essentiality, transposition feasibility), as described in Sec. 4.1 of the main paper. This factorization is intentional: phase priors encode cognitive tendencies, while feasibility modifiers prevent structurally invalid edits.

C.2. Correction Detection and Action Priors

PIE-V models correction generation as a two-stage process: (i) whether an error is noticed (detection that correlates with our Confusibility metric), and (ii) whether a corrective action is taken, with what latency and repair type. The base detectability term $b(\tau, \phi)$ in Eq. 7 of the main paper is a hand-specified prior over error type τ and phase bucket ϕ, designed to reflect cognitive regularities of error noticeability and recovery behavior (e.g., salient execution failures are more detectable than subtle deviations; detection probability decreases under high cognitive load) [16, 21, 24].

Base detectability table. We define $b(\tau, \phi)$ for each error type and phase bucket (EARLY, MID, LATE) and then modulate it multiplicatively with severity, essentiality, predicate salience, and load factors as in Eq. 7 of the main paper. These values are implementation priors (not learned parameters) and are fixed across all experiments.

Action, latency, and repair type. Conditioned on detection, PIE-V samples (i) whether the agent acts, (ii) correction latency in steps, and (iii) a correction type compatible with the triggering error (e.g., STOP_AND_FIX, REDO, ROLLBACK_AND_REDO, UNDO_EXTRA_STEP). This separation allows PIE-V to model both noticed-but-unfixed errors and explicit recovery traces.

C.3. Semantic Roles

PIE-V uses semantic step representations as a compact structural layer between free-form step text and the error simulator. Each step is represented as a predicate–argument expression of the form `PREDICATE(Role: value, Role: value, ...)` with nested structures when needed (e.g., locations, purposes, temporal clauses). These representations are used to (i) compute step complexity and phase boundaries, (ii) localize `WE` and role-level `S` edits, (iii) estimate role severity via a role-impact map, and (iv) provide predicate-conditioned role priors for selecting which

arguments to mutate.

Source of semantic role annotations. For the benchmark split, semantic representations are precomputed offline for the dataset step vocabulary and cached in a JSON mapping from step id to step_description and semantic_representation. We generate them with GPT-5.2 using a constrained prompt (Listing 1) and a strict JSON schema, in batches of step id → step description pairs, and require exact copying of input ids and step text in the output. The generation utility also normalizes step text and builds a reverse map for robust lookup across formatting variants (e.g., punctuation differences).

Listing 1. Excerpt of the prompt used to generate semantic representations (SemRep) for step descriptions.

```
You are a linguistic semantic analyzer.
For each procedural step description, generate
    the semantic representation in roles.

Hard constraints:
- Use compact single-line format:
    PREDICATE(Role: value, Role: value, ...)
- Predicates MUST be UPPERCASE.
- Prefer the role name Object (NOT Theme) for
    concrete manipulated entities.
- Use Agent: you unless another agent is
    explicitly stated.
- Prepositional phrases modifying a noun
    should be nested inside that noun.
- Keep entities as lowercase_with_underscores;
    nested structures are allowed.

Examples:
1) Insert the test swab into her nostril
INSERT(Agent: you, Object: test_swab,
    Destination: into(nostril(of(her))))

2) Add coffee grounds from a bowl to the
    filter in the French press
ADD(Agent: you, Object: coffee_grounds,
    Origin: bowl,
    Destination: filter(Location:
    in(french_press)))

3) Add cut onions to the egg in the mixing bowl
ADD(Agent: you, Object: cut(onions), Coobject:
    egg(Location: in(mixing_bowl)))

Return only JSON that matches the schema.
```

SemRep format and parsing. The semantic representation format is designed for controllable procedural editing rather than full semantic parsing. Predicates are uppercase action labels, and role values are compact entity expressions with optional nesting, as shown in Listing 2 (e.g., nested `Origin`, `Destination`, `Temporal`, `Purpose`, and `Result` structures). In the simulator, we use a shallow parser that extracts the main predicate and top-level role-value pairs from the representation string. This is sufficient for role-targeted mutation, essentiality heuristics, and local ordering guards without introducing a full symbolic world model.

Listing 2. Representative cached semantic representations (SemRep) used by PIE-V on the Ego-Exo4D split.

```
{
  "1": {
    "step_description": "Unbox covid test
    package",
    "semantic_representation": "UNBOX(Agent:
    you, Object: covid_test_package)"
  },
  "11": {
    "step_description": "Extract the test swab
    from her nostril",
    "semantic_representation": "EXTRACT(Agent:
    you, Object: test_swab, Origin:
    from(nostril(of(her))))"
  },
  "20": {
    "step_description": "Insert the collection
    swab in its pack for disposal",
    "semantic_representation": "INSERT(Agent:
    you, Object: collection_swab, Destination:
    in(pack(of(it))), Purpose: disposal)"
  },
  "21": {
    "step_description": "Cover the test vial
    with the lid",
    "semantic_representation": "COVER(Agent:
    you, Object: test_vial, Instrument: lid)"
  },
  "152": {
    "step_description": "Slowly backpedal the
    chain while applying the chain lube to
    each individual roller",
    "semantic_representation":
    "BACKPEDAL(Agent: you, Object: chain,
    Manner: slowly, Temporal:
    WHILE(APPLY(Agent: you, Object:
    chain_lube, Coobject:
    to(each_individual_roller(of(chain))))))"
  },
  "457": {
    "step_description": "Cut the sushi roll
    into smaller pieces on the cutting board",
    "semantic_representation": "CUT(Agent:
    you, Object: sushi_roll, Result:
    smaller_pieces, Location:
    on(cutting_board))"
  }
}
```

Role impact map (severity prior). PIE-V uses a role-impact map $\omega(r) \in \{\text{HIGH}, \text{MEDIUM}, \text{LOW}\}$ to control error severity in role-level edits. Impact labels are assigned manually based on linguistic and procedural semantics. Roles that typically determine the manipulated entity or a critical counterpart are treated as high impact, locative and instrumental roles are typically medium impact, and manner or temporal modifiers are typically low impact. Table 5

Table 5. Semantic roles, counts, and impact assignments for the 50-scenario Ego-Exo4D subset used in PIE-V. Counts are given in brackets.

Impact	Roles
High	Agent (1037), Object (1027), Coobject (51)
Medium	Location (288), Destination (281), Origin (259), Instrument (201), Purpose (62), Content (1)
Low	Manner (28), Temporal (18), Degree (11), Path (9), Duration (5), Direction (5), Proposition (5), Result (4), Quantity (3), Theme (2), Condition (2), Criterion (1)

summarizes the role inventory, counts, and impact assignments for the 50-scenario Ego-Exo4D subset used in this work. PIE-V uses this role-impact map both for role sampling and for deriving error severity from the set of mutated roles. Agent is excluded from mutation.

Predicate-conditioned role priors. To avoid uniformly random role edits, PIE-V uses predicate-conditioned role priors estimated from the semantic representation corpus. For each predicate, we aggregate the empirical frequency (or share) of roles observed with that predicate across the corpus and use the resulting distribution as $\mathrm{Prior}(r \mid \mathrm{pred})$. At generation time, for a step with predicate $\mathrm{pred}(a_t)$ and roles present in that step, the role score is proportional to

$$\mathrm{ImpactWeight}(\omega(r)) \cdot \big(0.2 + \mathrm{Prior}(r \mid \mathrm{pred}(a_t))\big),$$

where the additive constant provides smoothing for rare but valid roles. Role selection is restricted to roles present in the current step, and Agent is excluded from mutation. PIE-V occasionally may select two roles (instead of one) for compound WE events.

Auto-extension for generated steps. The LLM writer can introduce new step texts that are not present in the original vocabulary. To preserve SemRep-based validation and cascade checks, the writer pipeline supports automatic SemRep extension: unseen generated steps are resolved through a reverse text-to-id map and, if missing, are assigned new cached semantic representations via the same constrained GPT-5.2 SemRep generator.

C.4. Cascade Edits

Cascade edits are follow-up text rewrites that preserve procedure feasibility after a planned error changes an object, tool, or local state. They are produced in the LLM writer stage and validated in the LLM judge stage. In the metadata, cascade edits are marked as `mod="a"` and reuse the same error id `eid` as the triggering error. This makes the causal dependency explicit: the step is not a new mistake, but a downstream repair of textual consistency caused by an earlier mistake realization.

Cascade edits are necessary because PIE-V generates *coherent traces*, not isolated error labels. A locally valid error can make later steps impossible if references are left unchanged (e.g., a substituted object is never acquired, or a tool is removed before later use). The writer therefore rewrites downstream steps minimally, preserving the plan while keeping the procedure executable. The judge then checks that these adjustments are present when needed and that they remain linked to the same `eid`.

Example: GET-substitution propagation. In the Ego-Exo4D procedure `sfu_cooking_005_2`, PIE-V realizes a substitution at the early GET step by replacing *"Get cucumber from the refrigerator"* with *"Get bell pepper from the refrigerator"* (`eid=E01`). This change propagates to later steps that originally depend on cucumber. As a result, multiple downstream steps are rewritten as cascade edits (`mod="a"`, same `eid=E01`), i.e., the trace explicitly replaces cucumber-dependent steps with bell-pepper-dependent ones, including "Wash bell pepper with water", "Chop bell pepper with knife on the chopping board" (twice), and "Add chopped bell pepper into the bowl". Without these cascade edits, later steps would continue referring to cucumber even though the rewritten trace acquires bell pepper instead.

What is and is not a cascade edit. A cascade edit changes a later original step only as much as needed to restore consistency with an earlier error. It is not a primary error realization (`mod="e"`) and not a correction step (`mod="c"`). Primary error realizations instantiate the planned mistake, while cascade edits preserve executability and semantic continuity after that mistake.

Writer constraints for cascades. The writer prompt explicitly enforces cascade behavior when an error changes inventory or object identity, including a special rule for GET-like substitutions. A shortened excerpt is shown in Listing 3.

Listing 3. Writer prompt excerpt enforcing cascade edits after inventory-changing mistakes.

```
If the plan location/wording would make the
    sequence physically impossible, you must
    still realize the requested error type,
but choose the closest feasible variant near
    the same location (and then repair
    downstream references via cascade
    adjustments).

IMPORTANT SPECIAL CASE: GET-substitution
    propagation
```

```
If the planned error substitutes a GET-like
    step (get/take/pick/retrieve) so that you
    acquire Y instead of X,
then assume X is NOT available later unless it
    is explicitly acquired again (do NOT add
    new insertions unless the plan includes
    them).
Therefore, any later steps that refer to X
    should be cascade-adjusted to refer to Y:
- rewrite those downstream steps and mark them
    as mod="a" with the SAME eid as the
    GET-substitution.
This keeps the procedure executable while
    preserving the planned mistake.
```

D. Details on Experiments and Results

D.1. Prompt Logic and Structured Output Contracts

We do not reproduce the full LLM prompts here because they are long and implementation-specific. The full writer and judge prompts are available in the released codebase. Here, we summarize the core constraints they enforce and the structured output contracts that are required by the pipeline.

The writer prompt receives the original procedure, an error and correction plan, semantic representations of original steps, and ordering constraints. Its main objective is to realize the planned mistakes while keeping the rewritten procedure physically feasible and executable. The prompt explicitly requires structured JSON output with a rewritten step list (`final_steps`) and a timeline mapping (`meta`) that marks unchanged steps, primary error realizations, cascade edits, insertions, deletions, corrections, and transposition pairs.

The judge prompt validates plan compliance and procedural coherence and proposes minimal repairs when needed. In particular, it checks that each planned error and correction id is realized, that transpositions are encoded via the required `ms`/`mt` pair, and that downstream references are repaired through cascade edits when an earlier error changes object or tool availability. These prompt-level constraints are combined with deterministic checks in the pipeline, so acceptance depends on both LLM reasoning and rule-based validation. Representative prompt fragments for cascade constraints and video style locking are shown in Listings 3 and 5.

For comparison, Listing 4 shows the freeform baseline prompt used to generate mistake-aware procedures without structured planning.

Listing 4. Freeform LLM baseline prompt for direct mistake-aware procedure rewriting (without PIE-V planning priors).

```
instructions = (
    f"### ROLE: Procedure Editor for
    '{scenario}'\n"
    f"You will receive a step-by-step
    procedure.\n"
    f"Your task is to produce an edited final
    procedure that contains 1 to 3 plausible
    human mistakes.\n"
    f"You may also add 0 to 2 explicit
    correction steps.\n\n"
    f"### ERROR TYPES (HIGH-LEVEL)\n"
    f"- wrong_execution (mod='we'): Keep the
    same general goal, but execute it slightly
    wrong (wrong amount, messy action, wrong
    orientation).\n"
    f"- substitution (mod='s'): Replace the
    step with a different plausible action
    caused by confusion.\n"
    f"  Do not copy a later step verbatim.\n"
    f"- insertion (mod='i'): Insert one extra
    plausible step (unnecessary repetition,
    extra cleaning, checking, etc.).\n"
    f"- deletion: Remove the step completely
    (do not mention it was skipped). Add it to
    'del'.\n"
    f"- transposition: Swap two steps.\n"
    f"  Use mod='ms' for the moved SOURCE step
    and mod='mt' for the moved TARGET step
    (both verbatim text, only order
    changes).\n\n"
    f"### HARD CONSTRAINTS\n"
    f"1) VERBATIM PRESERVATION: Keep most
    steps unchanged unless directly edited.\n"
    f"2) IMPERATIVE STYLE: Use direct
    imperative commands. No story.\n"
    f"3) SOURCE INDEX RANGE: Every meta
    source_idx MUST be a valid original index
    in [0, {len(steps)-1}]. Never use -1.\n"
    f"4) PHYSICAL PLAUSIBILITY: Do not use
    tools/ingredients/objects before they
    appear earlier in the procedure.\n"
    f"5) METADATA ALIGNMENT: 'final_steps' and
    'meta' must have the exact same
    length.\n\n"
    f"6) UNCHANGED MEANS VERBATIM: If mod='u',
    the final step text MUST match the
    original step text exactly.\n"
    f"7) TRANSPOSITION RULE: If using
    transposition, error_id must be non-null
    and exactly TWO steps must share that
    error_id:\n"
    f"   - one with mod='ms' and one with
    mod='mt'.\n\n"
    f"8) NO FAKE ERRORS: If mod in ['we','s'],
    the text MUST be meaningfully different
    from the original step at source_idx.\n"
    f"9) MOVE IS VERBATIM: If mod in
    ['ms','mt'], the text MUST be identical to
    the original step at source_idx (only
    moved).\n"
    f"10) INSERTION IS NEW: If mod='i', the
    inserted text MUST NOT be identical to the
    anchor step at source_idx.\n"
    f"11) CORRECTION NEEDS ID: If mod='c',
    correction_id must be non-null like 'C01'
    and the text must be new.\n"
    f"12) ERROR IDS: For mod in
    ['we','s','i','ms','mt'] use error_id like
    'E01'. For deletions also.\n"
    f"### OUTPUT FORMAT (STRICT JSON ONLY)\n"
```

```
    f"- final_steps: list[str]\n"
    f"- meta: list[list], one per final step:
    [source_idx, mod, error_id,
    correction_id]\n"
    f"  mod_type: 'u'(unchanged),
    'we'(wrong_execution), 's'(substitution),
    'ms'(moved_source), 'mt'(moved_target),
    'c'(correction), 'i'(insertion)\n"
    f"- del: list[list] for deleted steps:
    [source_idx, error_id]\n\n"
    f"For every deletion, error_id must be a
    string like 'E01' (never null).\n"
    f"Return ONLY a single JSON object. No
    markdown. No extra text.\n"
    f"Do NOT repeat the prompt. Output ONLY
    JSON.\n"
)
```

D.2. Annotation Data Processing and Aggregation

This subsection describes how raw annotator responses are converted into the aggregate statistics reported in the main paper tables.

Metric types. Our rubric mixes categorical judgments (e.g., Error Validity, Procedure Logic, State-Change Coherence, Taxonomy Fit) and Likert-scale ratings (e.g., Human Plausibility, Confusability, Sequence Consistency, Video Plausibility, Text–Video Grounding). This is intentional, because the rubric is designed to capture both relatively stable categorizations and subjective human judgments about realism and noticeability.

Error Validity. Error Validity is annotated as a binary judgment (Yes/No), namely whether the highlighted deviation is a consequential procedural error under our definition. In the main-paper tables, we report the percentage of "Yes" judgments aggregated at the sample level. Disagreements on this metric therefore reflect ambiguity in the generated behavior (or dataset annotation), not a multi-level scoring scheme. In practice, disagreement on this binary metric often comes from freeform outputs that are lexically marked as "accidental" but have weak procedural consequences. We provide representative examples in Sec. D.4.

Procedure Logic with confidence weighting. Procedure Logic is annotated as a binary judgment together with confidence; aggregation follows the confidence-weighted formulation defined in the main paper (Sec. 3).

Likert-scale aggregation. For Human Plausibility, Confusability, Sequence Consistency, Video Plausibility, and Text–Video Grounding, we report the arithmetic mean over annotator ratings.

Agreement. We compute Krippendorff's α separately for each metric and setting. Higher agreement is expected for Error Validity and Taxonomy Fit, while lower agreement on Human Plausibility and Confusability reflects genuine variation in human judgments about naturalness and noticeability.

D.3. Additional Breakdowns for Ego-Exo4D Generations

Table 4 in the main paper reports scale statistics for the four Ego-Exo4D generation settings. Two patterns are consistent across the audited outputs. First, freeform generation underproduces mistakes relative to PIE-V+Judge settings, with fewer mistake steps and a lower average number of mistakes per video. Second, even when freeform outputs are fluent at the sentence level, they more often fail to realize clearly consequential deviations or to preserve long-horizon procedural consistency, which is reflected in the rubric aggregates in Table 3 of the main paper.

The freeform models also show different error-type biases. For Qwen freeform, the generated distribution is strongly skewed toward deletions (Deletion 31, Substitution 12, Transposition 5, Wrong Execution 5, Insertion 2, Correction 1), and qualitative inspection shows many weak or under-realized edits. For GPT freeform, the distribution is more concentrated on Wrong Execution and Substitution (Wrong Execution 27, Substitution 13, Transposition 8, Insertion 2, Correction 4, Deletion 0), which often produces locally plausible deviations but still under-produces explicit recovery behavior relative to PIE-V. These tendencies motivate the use of explicit planning, cascade edits, and judge-side validation in PIE-V.

D.4. Qualitative Failure Cases of Freeform Baselines

Binary Error Validity disagreement from weakly consequential freeform insertion (`indiana_bike_03_1`). A recurring source of disagreement on Error Validity in freeform generations is that the LLM marks an event as accidental without introducing a clearly consequential procedural failure. For example, in a bike-chain cleaning procedure, GPT inserts *"Accidentally knock the chain lube bottle over on the floor while reaching for it"* between *"Hold the toothbrushes to the chain as you backpedal with your other hand"* and *"Get the chain lube from the floor"*. Some annotators judge this as a procedural error, while others judge it as a harmless deviation because the procedure remains executable and the intended outcome is not meaningfully affected. This is precisely the kind of boundary case that Error Validity is designed to expose.

Qwen repetition instead of a meaningful substitution (`nus_covidtest_15_2`). In a COVID-test procedure,

a planned substitution near the disposal and waiting stage should replace the step *"Arrange test materials in the plastic bag for disposal"*. Qwen, even with judge, realizes the deviation by duplicating the previous waiting step, producing two consecutive occurrences of *"Wait for a few minutes"*. This keeps the text locally fluent but weakens procedural specificity and does not create a clear, interpretable mistake mechanism.

By contrast, GPT produces a more consequential and traceable deviation in the same region, for example *"Dispose the test plate into the plastic bag"*, followed by a compensating step *"Takes the test plate from the plastic bag to check the test plate for the results"*. Although still imperfect, this sequence preserves a clearer error and recovery interpretation for benchmarking.

More broadly, freeform outputs often use lexical markers such as *"Accidentally"* or *"Stop and ..."* while describing behavior that is only weakly harmful, visually subtle, or procedurally neutral. This is one reason Error Validity remains an informative metric for freeform baselines: it measures whether the generated deviation is perceived as a consequential procedural error, not whether the text merely sounds like one.

D.5. From LLM Step Text to Video Generation Prompts

A practical scalability bottleneck of PIE-V is the conversion of LLM-generated step text into video-generation prompts. Writer and judge outputs are optimized for procedural coherence and annotation traceability, not for direct video synthesis. As a result, many generated step descriptions require manual prompt compilation to make the intended visual event physically explicit and compatible with a specific video model.

This issue is especially visible in freeform generations. LLMs often produce text that is linguistically marked as an error, for example with words such as *"accidentally"*, while the described behavior is visually subtle, weakly consequential, or underspecified for video generation. Similarly, correction steps may be described as meta-actions, for example *"Stop and fix..."*, that require rewriting into concrete visible behavior before synthesis.

In our workflow, each edited segment is therefore paired with a model-specific prompt that specifies observable actions, object identity, camera constraints, and scene continuity. This prompt compilation step is currently the main scalability bottleneck of the video stage in PIE-V.

Listing 5. Example style-lock prompt used for egocentric video generation (bike-repair scenario).

```
PROMPT_STYLE_LOCK = (
    "Egocentric head-mounted camera, fixed
    POV, same fisheye lens and same dark
    circular vignette. "
    "Same bike repair workshop, same bike
    wheel and tools, same hands and body, same
    lighting. "
    "No new objects, no swaps, no text
    overlays, no reframing, no cut."
)
```

For Ego-Exo4D edits, a short style-lock prefix is often sufficient to preserve egocentric viewpoint and scene identity across regenerated clips (Listing 5).

D.6. Video Model Constraints and Editing Windows

Video generation models differ in conditioning interface and clip-length limits. Some support text-only generation, while others require boundary anchors, for example start and end frames, or support only specific durations. Because of these constraints, long procedural steps cannot always be replaced in a single pass and often require windowing, bridge clips, and stitching.

In PIE-V, edited windows are selected around the targeted mistake or correction, while unchanged parts of the episode are preserved from the original video. When a full-step replacement is not feasible, we generate shorter clips and reconnect them with boundary-aware splicing and transition smoothing. This is particularly important for egocentric video, where continuity of hands, tools, and camera viewpoint strongly affects plausibility.

Different video models also require different conditioning inputs, including text-only mode, boundary-frame mode, and model-specific editing interfaces, which prevents a single universal editing script. We also tested prompt-based editing on non-egocentric videos, but support is uneven across models, especially for clips with clearly visible people, which makes the current workflow more reliable on egocentric footage.

D.7. Limitations and Future Directions

PIE-V is intended as a controlled framework for constructing and auditing mistake-aware procedural traces, and the current version should be understood as a first step rather than a final fully scaled benchmark.

Benchmark scale and coverage. The current benchmark remains modest in scale: it is built from 50 Ego-Exo4D scenarios and contains 102 injected mistakes and 27 recovery corrections. This is sufficient for an initial controlled study, but it does not yet support strong claims of exhaustive coverage over mistake diversity, recovery patterns, or domain variation. In particular, some error types, correction strategies, and long-horizon dependencies are still underrepresented. Expanding the benchmark with more tasks, multiple variants per scenario, and broader procedural domains is a natural next step.

Downstream utility is not yet established. The present paper evaluates PIE-V primarily through human judgment and comparative auditing. While this is appropriate for validating plausibility and coherence, we do not yet show that PIE-V data improves downstream models for mistake detection, correction prediction, or post-completion verification. Demonstrating such gains through controlled training and transfer experiments is an important direction for future work.

The video stage is not yet fully automated, but it already changes the cost profile of mistake-aware data construction. The textual planning and rewriting components are substantially more scalable than the current video-editing stage. In practice, converting rewritten procedural steps into high-quality video-generation prompts still requires model-specific prompt compilation and manual iteration. This bottleneck is compounded by heterogeneous video-model interfaces, conditioning requirements, and clip-length limits, which prevent a single universal editing pipeline.

At the same time, PIE-V operates under a fundamentally different data-construction paradigm from existing mistake-aware procedural video datasets, including the ones we audit, which rely on newly recorded human executions of erroneous procedures. Such collection typically requires participant time, repeated performances, annotation effort, and often specialized capture setups. By contrast, PIE-V starts from existing clean procedural videos and edits only targeted segments. In this sense, even in its current partially manual form, PIE-V already offers a practical and resource-efficient alternative to full re-recording, reducing both collection cost and human effort while preserving control and auditability.

To our knowledge, PIE-V is also among the first mistake-aware procedural benchmark constructions to rely on prompt-based editing of existing video segments rather than new recordings of erroneous executions. We therefore view the current pipeline not only as a benchmark-generation method, but also as an early demonstration of a different and potentially much cheaper way to build mistake-aware procedural video resources.

We also expect the scalability of this stage to improve as prompt-based and instruction-guided video editing models continue to advance [11, 13, 14]. Recent progress in general video-to-video editing, instruction-based video editing, and emerging egocentric video editing suggests that more automated and temporally consistent editing pipelines may substantially reduce the need for manual prompt engineering. However, adapting such models to controlled procedural mistake construction remains a separate research problem.

Video-side evaluation is necessarily selective. A limitation of the current study is that Video Plausibility and Text–Video Grounding are not reported for every generation setting. Producing and auditing fully edited videos is substantially more resource-intensive than text-only evaluation, since each setting requires video generation, clip selection, temporal stitching, and additional human assessment of the resulting outputs. For this reason, we adopted a staged evaluation design: generation settings were first compared at the textual/procedural level, and only the strongest configuration was carried forward to the full video stage. We view this as an intentional and pragmatic use of limited computational and annotation resources rather than as an arbitrary omission, because scaling clearly weaker text-generation settings to costly video realization would add substantial expense with limited scientific value. A broader cross-setting video evaluation remains an important direction for future work, but it would require substantially greater compute, annotation time, and human effort.

Dependence on source data and external models. The current benchmark is derived from Ego-Exo4D keysteps and therefore inherits both the strengths and the limitations of that source representation. More broadly, PIE-V also depends on rapidly evolving LLM and video-generation models whose behavior, interfaces, and output quality may change over time. For this reason, the current results should be interpreted as evidence for the usefulness of the PIE-V design principles and evaluation protocol, rather than as a claim that one fixed generated benchmark is final or universal.

Generated mistakes remain approximations of human behavior. Even when errors are psychology-informed, role-constrained, and judged coherent by annotators, generated traces remain approximations rather than direct observations of naturally occurring human mistakes. They may miss social context, tacit goals, embodied variability, and opportunistic recovery strategies that arise in real-world execution. We therefore view PIE-V as complementary to naturally observed mistake datasets rather than a replacement for them.

References

[1] Siddhant Bansal, Chetan Arora, and C. V. Jawahar. My view is the best view: Procedure learning from egocentric videos, 2022. 2

[2] Michael D. Byrne and Susan Bovair. A working memory model of a common procedural error. *Cogn. Sci.*, 21:31–61, 1997. 4

[3] Michael D. Byrne and Elizabeth M. Davis. Task structure and postcompletion error in the execution of a routine procedure. *Human Factors*, 48(4):627–638, 2006. 4

[4] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Sanja Fidler, Antonino Furnari, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, and Michael Wray. The epic-kitchens dataset: Collection, challenges and baselines, 2020. 2

[5] Guodong Ding, Fadime Sener, Shugao Ma, and Angela Yao. Every mistake counts in assembly. *ArXiv*, abs/2307.16453, 2023. 1

[6] Reza Ghoddoosian, Isht Dwivedi, Nakul Agarwal, and Behzad Dariush. Weakly-supervised action segmentation and unseen error detection in anomalous instructional videos. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10094–10104, 2023. 1, 2

[7] Kristen Grauman, Andrew Westbury, Lorenzo Torresani, Kris Kitani, Jitendra Malik, Triantafyllos Afouras, Kumar Ashutosh, Vijay Baiyya, Siddhant Bansal, Bikram Boote, Eugene Byrne, Zach Chavis, Joya Chen, Feng Cheng, Fu-Jen Chu, Sean Crane, Avijit Dasgupta, Jing Dong, Maria Escobar, Cristhian Forigua, Abrham Gebreselasie, Sanjay Haresh, Jing Huang, Md Mohaiminul Islam, Suyog Jain, Rawal Khirodkar, Devansh Kukreja, Kevin J Liang, Jia-Wei Liu, Sagnik Majumder, Yongsen Mao, Miguel Martin, Effrosyni Mavroudi, Tushar Nagarajan, Francesco Ragusa, Santhosh Kumar Ramakrishnan, Luigi Seminara, Arjun Somayazulu, Yale Song, Shan Su, Zihui Xue, Edward Zhang, Jinxu Zhang, Angela Castillo, Changan Chen, Xinzhu Fu, Ryosuke Furuta, Cristina Gonzalez, Prince Gupta, Jiabo Hu, Yifei Huang, Yiming Huang, Weslie Khoo, Anush Kumar, Robert Kuo, Sach Lakhavani, Miao Liu, Mi Luo, Zhengyi Luo, Brighid Meredith, Austin Miller, Oluwatumininu Oguntola, Xiaqing Pan, Penny Peng, Shraman Pramanick, Merey Ramazanova, Fiona Ryan, Wei Shan, Kiran Somasundaram, Chenan Song, Audrey Southerland, Masatoshi Tateno, Huiyu Wang, Yuchen Wang, Takuma Yagi, Mingfei Yan, Xitong Yang, Zecheng Yu, Shengxin Cindy Zha, Chen Zhao, Ziwei Zhao, Zhifan Zhu, Jeff Zhuo, Pablo Arbelaez, Gedas Bertasius, David Crandall, Dima Damen, Jakob Engel, Giovanni Maria Farinella, Antonino Furnari, Bernard Ghanem, Judy Hoffman, C. V. Jawahar, Richard Newcombe, Hyun Soo Park, James M. Rehg, Yoichi Sato, Manolis Savva, Jianbo Shi, Mike Zheng Shou, and Michael Wray. Ego-exo4d: Understanding skilled human activity from first- and third-person perspectives, 2024. 2, 3

[8] Yuto Haneji, Taichi Nishimura, Hirotaka Kameko, Keisuke Shirai, Tomoya Yoshida, Keiya Kajimura, Koki Yamamoto, Taiyu Cui, Tomohiro Nishimoto, and Shinsuke Mori. Egooops: A dataset for mistake action detection from egocentric videos referring to procedural texts, 2025. 1, 2, 3

[9] Youngkyoon Jang, Brian T. Sullivan, Casimir J. H. Ludwig, Iain D. Gilchrist, Dima Damen, and W. Mayol-Cuevas. Epic-tent: An egocentric video dataset for camping tent assembly. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 4461–4469, 2019. 2

[10] Marcel Adam Just and Patricia A. Carpenter. A capacity theory of comprehension: individual differences in working memory. *Psychological review*, 99 1:122–49, 1992. 4

[11] Max Ku, Cong Wei, Weiming Ren, Harry Yang, and Wenhu Chen. Anyv2v: A tuning-free framework for any video-to-video editing tasks, 2024. 11

[12] 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 (CVPR)*, pages 18655–18666, 2024. 1, 2, 3

[13] Runjia Li, Moayed Haji-Ali, Ashkan Mirzaei, Chaoyang Wang, Arpit Sahni, Ivan Skorokhodov, Aliaksandr Siarohin, Tomas Jakab, Junlin Han, Sergey Tulyakov, Philip Torr, and Willi Menapace. Egoedit: Dataset, real-time streaming model, and benchmark for egocentric video editing, 2025. 11

[14] Jinjie Mai, Chaoyang Wang, Guocheng Gordon Qian, Willi Menapace, Sergey Tulyakov, Bernard Ghanem, Peter Wonka, and Ashkan Mirzaei. Easyv2v: A high-quality instruction-based video editing framework, 2025. 11

[15] Stephen McKenna and Sebastian Stein. 50 salads. https://discovery.dundee.ac.uk/en/datasets/50-salads/, 2012. 2

[16] Donald A. Norman. *The Psychology of Everyday Things*. Basic Books, New York, 1988. 4, 5

[17] Fred Paas, Alexander Renkl, and John Sweller. Cognitive load theory and instructional design: Recent developments. *Educational Psychologist*, 38:1 – 4, 2003. 4

[18] Rohith Peddi, Shivvrat Arya, Bharath Challa, Likhitha Pallapothula, Akshay Vyas, Bhavya Gouripeddi, Jikai Wang, Qifan Zhang, Vasundhara Komaragiri, Eric Ragan, Nicholas Ruozzi, Yu Xiang, and Vibhav Gogate. Captaincook4d: A dataset for understanding errors in procedural activities, 2024. 1, 2

[19] Yicheng Qian, Weixin Luo, Dongze Lian, Xu Tang, Peilin Zhao, and Shenghua Gao. Svip: Sequence verification for procedures in videos, 2022. 1, 2

[20] Francesco Ragusa, Antonino Furnari, Salvatore Livatino, and Giovanni Maria Farinella. The meccano dataset: Understanding human-object interactions from egocentric videos in an industrial-like domain, 2020. 1, 2

[21] James Reason. *Human Error*. Cambridge University Press, 1990. 4, 5

[22] Tim J. Schoonbeek, Tim Houben, Hans Onvlee, Peter H. N. de With, and Fons van der Sommen. Industreal: A dataset for procedure step recognition handling execution errors in egocentric videos in an industrial-like setting, 2023. 1, 2

[23] Fadime Sener, Dibyadip Chatterjee, Daniel Shelepov, Kun He, Dipika Singhania, Robert Wang, and Angela Yao. Assembly101: A large-scale multi-view video dataset for understanding procedural activities. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21064–21074, 2022. 1, 2

[24] Franklin P. Tamborello and J. Gregory Trafton. A long-term memory competitive process model of a common procedural error. *Cognitive Science*, 35, 2013. 5

[25] Xin Wang, Taein Kwon, Mahdi Rad, Bowen Pan, Ishani Chakraborty, Sean Andrist, D. Bohus, Ashley Feniello, Bugra Tekin, F. Frujeri, Neel Joshi, and Marc Pollefeys.

Holoassist: an egocentric human interaction dataset for interactive ai assistants in the real world. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 20213–20224, 2023. 1, 2