Do You See What I Am Pointing At? Gesture-Based Egocentric Video Question Answering

Supplementary Material

Backbone	r	a	lr	Resolution
InternVL3-8B	64	128	1e-5	448×448
InternVL3-14B	32	64	2e-5	448×448
LLaVA-OneVision-7B	32	64	1e-5	384×384

Table 7. **Hyperparameters for HINT per backbone.** This table summarizes the optimized training configurations. Here, 'r' denotes the LoRA rank, 'a' the LoRA scaling factor, and 'lr' the learning rate used during finetuning.

A. Implementation Details

A.1. HINT Training Details

Table 7 summarizes the optimized hyperparameters used to finetune HINT on each backbone. We initialized the hyperparameter search around values commonly used in prior works and performed a grid search over LoRA rank and scaling factor pairs of {(8, 16), (16, 32), (32, 64), (64, 128)} and learning rates in {2e-7, 1e-5, 2e-5}. The reported configurations were selected based on the best validation performance for each backbone.

In all experiments, we finetune only the LoRA adapters and the keypoint adapter while freezing the remaining backbone parameters. We use AdamW as the optimizer with a cosine learning rate schedule and a linear warm-up ratio of 0.03. Input videos are uniformly sampled in time to a fixed number of 32 frames, resized to the resolution in Tab. 7, and normalized following each backbone's default preprocessing.

A.2. Evaluation Protocol

We evaluate all models using multiple-choice accuracy. Each example consists of a question, candidate options, and the index of the correct option. Models are instructed to answer by selecting an option letter ("A", "B", etc.). We handle several common output formats. If the entire string (after trimming whitespace) is a single option letter, we take it as the prediction. Otherwise, we progressively clean the output by stripping special markup such as `<answer>...</answer>`, as well as trailing control tokens (e.g., `<|im_end|>`) and boilerplate phrases (e.g., "Here's the answer:"). After this cleaning step, we apply a sequence of regular expressions that search for: (i) a letter enclosed in parentheses, e.g., "(A)"; (ii) a letter followed by punctuation, e.g., "A.", "B)" or "C]"; and (iii) an "Answer:" pattern at the end of the string, e.g., "Answer: D". In all cases, we restrict matches to letters between "A" and the last valid option letter for that specific example. If a match is found and the letter is valid, we treat it as the model's prediction and otherwise, we mark the prediction as invalid. Multiple-choice accuracy is computed as the fraction of examples for which the extracted option letter exactly matches the ground-truth option. For all baselines and HINT variants reported in the paper, we successfully extracted a valid option letter for every prediction, so the reported accuracies correspond to exact letter-wise matches without any manual corrections or post-hoc filtering.

Method	Video-MME	MVBench	EgoSchema	EgoBlind
InternVL3-8B	64.2	73.2	67.2	52.8
HINT $_{\text{InternVL3-8B}}$	64.6	73.2	67.1	57.5

Table 8. **Performance on existing video understanding benchmarks.** Comparison of HINT against the baseline InternVL3-8B on Video-MME [15], MVBench [26], EgoSchema [28] (MCQ), and EgoBlind [11]. The results indicate that finetuning on our dataset preserves the model's general video understanding capabilities.

B. Additional Analysis

B.1. HINT Performance on Existing Benchmarks

In Table 8, we report the performance of HINT on standard video understanding benchmarks to investigate whether finetuning on our proposed dataset compromises the model's pretrained general capabilities. We compare our method with the backbone model, InternVL3-8B, on Video-MME [15], MVBench [26], and EgoSchema [28]. All evaluations use 32 uniformly sampled frames without any additional fine-tuning on the target benchmarks. This result demonstrates that the gesture-aware representations learned by HINT transfer positively to related egocentric assistive QA scenarios, even when explicit pointing gestures are absent in the target benchmark. As can be easily observed, HINT achieves comparable performance compared to the baseline. This demonstrates that our finetuning strategy effectively injects pointing gesture understanding capability without leading to catastrophic forgetting in general video understanding tasks, and also can be transfer positively to related egocentric assistive QA scenarios even when explicit pointing gestures are absent in the target benchmark.

B.2. Human Performance

We report the average accuracy of 5 participants. Each participant evaluated the full test set. The near-ceiling performance (total average 95.9%) confirms the questions are clear and easy for humans, yet reveals a substantial gap to

	Ref.	Temp.	Spat.	Count	Attr.	Feed.
Human	98.2	94.3	93.3	92.8	96.5	100.0

Table 9. **Human performance on EGOPOINTVQA.** We report the average accuracy of 5 human participants, each evaluating the full test set (672 questions).

Video	Reference	Temporal	Spatial	Attribute
Original	75.0	66.1	64.9	61.0
w/o Hand	41.7	21.4	44.4	36.3

Table 10. **Ablation study on the effect of hand gestures.** Performance comparison on our dataset with and without the pointing hand visible in the video frames. The significant drop in performance across all tasks in the 'w/o Hand' setting confirms that the pointing gesture is essential for identifying the target.

	Reference	Temporal	Spatial	Count	Attribute	Feedback
Random	20.0	20.0	27.0	20.0	20.0	50.0
Blind	16.7	25.5	32.0	6.3	22.5	54.9
Choices-only	20.8	25.5	20.8	10.4	21.6	54.1

Table 11. **Dataset bias analysis.** We evaluate two text-only baselines to detect potential shortcuts: "Blind" receives the question text without video, and "Choices-only" receives only the answer options without the question or video. Performance near random chance confirms that EGOPOINTVQA requires visual grounding.

current MLLMs (best performing model 68.1%).

B.3. Effect of Hand Gestures in Video

To assess the importance of explicit pointing cues, we construct a static-video variant of EGOPOINTVQA in which the camera wearer's hand is removed while keeping the rest of the scene and camera motion unchanged. We then evaluate models, including HINT, on this modified data using the same training and evaluation protocol as in the main experiments. Table 10 shows that removing hand gestures causes a large degradation in performance. This confirms that egocentric pointing cues are critical for resolving deictic references in EGOPOINTVQA.

B.4. Dataset Bias Analysis

To verify that EGOPOINTVQA does not contain unintended textual shortcuts, we evaluate two degenerate baselines that receive no visual input (Table 11). The **Blind** baseline feeds only the question text (without the video) to the model, testing whether the question wording alone leaks the answer. The **Choices-only** baseline provides only the multiple-choice options (without either the question or video), testing whether the answer can be guessed from the option distribution. Both baselines perform near random chance across all six task categories: Blind achieves 16.7–54.9% and Choices-only achieves 10.4–54.1%, closely matching the random baseline of 20.0–50.0%. These results confirm that EGOPOINTVQA genuinely requires visual grounding of pointing gestures to resolve deictic references, and that our question-answer generation pipeline does not introduce systematic textual or statistical shortcuts.

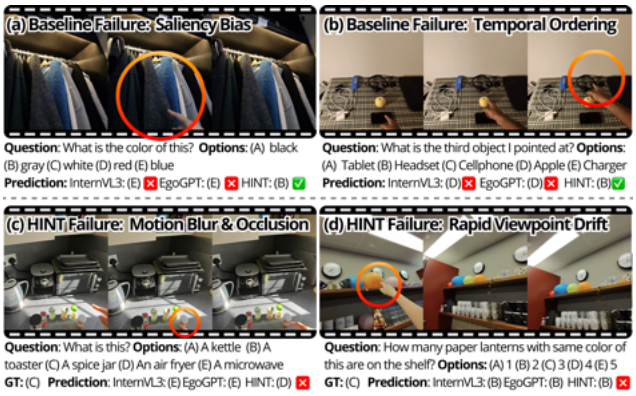

Figure 7. **Representative failure cases on EGOPOINTVQA.** (a)-(b): baseline MLLM failures due to saliency bias and temporal confusion, respectively. (c)-(d): remaining HINT failures caused by unreliable hand keypoints and rapid viewpoint drift.

B.5. Failure Analysis

In Figure 7, we visualize example failure modes to better understand the limitations of both existing MLLMs and our proposed HINT.

Existing MLLM failure modes. We identify two dominant error patterns in baseline models. (a) *Saliency/center bias*: in cluttered scenes, models tend to predict a visually prominent or centrally located object rather than the one actually being pointed at. For example, the model may focus on a nearby blue shirt instead of the actual referent. (b) *Temporal confusion*: when multiple objects are sequentially pointed at, baseline models frequently confuse the temporal order, e.g., predicting the second pointed object when the question asks about the first. HINT mitigates both failure modes by providing frame-aligned 3D hand geometry that explicitly anchors deictic references to the correct spatial and temporal locations.

Remaining HINT failure modes. Despite the overall improvements, HINT fails under certain challenging conditions. (c) *Unreliable gesture signal*: when the hand is affected by motion blur or partial occlusion, the 3D hand reconstruction from WiLoR becomes noisy. In such cases, the hand detection confidence may fall below the threshold τ, resulting in absent hand intent tokens, or the tokens may encode misleading pose information. (d) *Rapid viewpoint drift*: fast head movements can cause the target object to leave the camera's field of view entirely, making it harder for the model to associate the gesture with the correct referent regardless of the quality of hand tokens. These remaining errors are largely attributable to input signal quality rather than architectural limitations, suggesting that advances in robust hand pose estimation and temporal object tracking under egocentric motion would yield further gains.

C. EGOPOINTVQA

In this section, we provide additional analysis of EGOPOINTVQA, qualitative visualizations, and a detailed description of the question answer pair generation pipeline.

C.1. Video Collection Details

Participants. We recruit 20 participants from 12 nationalities (11 female, 9 male; ages 18–45) to collect the real-world portion of EGOPOINTVQA. The diverse participant pool ensures variation in hand shape, skin tone, and pointing style, which encourages models to generalize across different users rather than overfitting to a narrow demographic.

Scenes and activities. Real-world videos are captured across a broad variety of settings, including offices, kitchens, living rooms, streets, train platforms, and balconies. Activities during recording span desk work, organizing, artwork, and cooking, reflecting natural everyday scenarios in which pointing gestures commonly occur. Of the 400 collected videos, 360 are recorded in indoor environments and 40 in outdoor settings.

Capture protocol. Each participant wears Meta Ray-Ban smart glasses and is instructed to point with one hand using an extended index finger for at least 2 seconds at objects of their choice. We require that more than 3 objects are visible in the scene to ensure sufficient visual complexity for generating challenging deictic questions. Each clip ranges from 3–8 seconds at 30 FPS with a resolution of 1536×2048.

Data Split	# Vid.	# QA	Vid. Dur.(s)	# Obj.
Train(Real)	100	640	4.61	22.5
Train(Synthetic)	4,000	18,745	11.6	11.6
Test (Real)	300	672	5.05	16.5

Table 12. **EGOPOINTVQA statistics.** Statistics of the dataset across synthetic and real-world splits. Real-world clips generally feature higher object density (scene complexity) than synthetic clips.

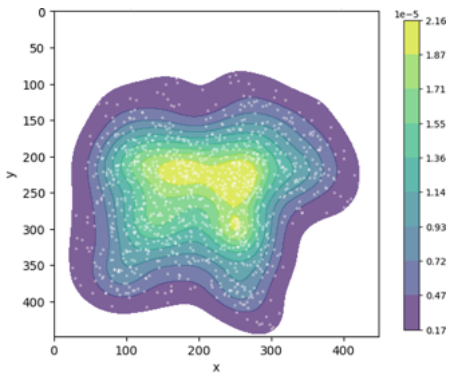

Figure 8. **Spatial distribution of target objects in EGOPOINTVQA.** 'x' and 'y' axes refer to the horizontal and vertical pixel coordinates, respectively. Individual object locations are shown as scattered dots.

C.2. Dataset Statistics

Table 12 summarizes the scale and complexity of EGOPOINTVQA. The dataset is composed of a large-scale synthetic training set (4,000 videos) to encourage robust generalization, complemented by real-world data for domain adaptation and testing. Notably, the real-world videos contain a higher density of objects (avg. 22.5 and 16.5 per video) compared to the synthetic set (avg. 11.6 per video), presenting a greater challenge in grounding target pointed objects.

In Fig. 8, we visualize the spatial distribution of target objects by their center coordinate within the frame. While there is a natural center bias typical of egocentric videos, where users tend to center objects they interact with, the heatmap demonstrates a diverse distribution of the object location. This broad distribution confirms that models cannot rely solely on center priors and must utilize the pointing gestures to resolve references.

C.3. Extended Visualization of EGOPOINTVQA

We provide additional qualitative examples from the real-world split of EGOPOINTVQA in Figures 19 and 20. Figure 19 presents samples recorded in **indoor environments**, illustrating various household objects and close-range pointing gestures. In contrast, Figure 20 depicts sequences captured in **outdoor settings**. These visualizations showcase the range of scenes and contexts included in the collected data.

C.4. Synthetic Video Generation

We generate synthetic egocentric videos using the AI2-THOR simulator [23] running in Unity. Our implementation utilizes AI2-THOR commit `0+8524ea` and Unity version `2020.3.25f1` The video generation process follows four steps.

First, we randomly sample a scene, an agent location, and a head camera orientation. We keep only viewpoints where at least three objects are simultaneously visible according to ground-truth instance segmentation masks from the simulator.

From the visible entities, we select a target object of the pointing gesture. To ensure valid grounding, we require the object to meet two criteria: more than 10% of its pixels must be visible, and its depth must be within 2.0 meters from the agent.

For the target object, we retrieve a pointing animation from the MIXAMO [21] library that possesses the most similar initial pose to the required trajectory. We then apply Inverse Kinematics (IK) to refine the avatar's arm and hand configuration, forcing the index finger to align precisely with the target object.

After IK adjustment, we verify that the pointing ray originating at the fingertip intersects the 3D bounding box of

the target. If the pointing deviates or misses the target, we discard this sample and resample a new viewpoint/pose.

C.5. Automatic Question and Answer Generation Details

We provide a detailed breakdown of the prompts and logic used in our three-stage generation pipeline described in the main paper (§ 3.3). We utilize InternVL3-78B [53] for scene information extraction and multiple-choice question-answer pair generation (Stages 1& 2) and GPT-4o [20] for linguistic refinement and quality control (Stage 3).

Stage 1: Dense scene information extraction. Instead of generic captions, we require structured scene graphs to ground the subsequent QA generation. As shown in Fig. 9, we prompt the annotator MLLM to act as a Dense Scene Fact Miner." The model analyzes the video frames and outputs a JSON list of all salient objects. To ensure high-quality grounding, we enforce two strict constraints in the prompt: (1) Hand-Agnostic Descriptions: The model is explicitly forbidden from describing human interaction (e.g., pointed by a hand) to ensure object attributes are described objectively based on their visual state. (2) Discriminative Expressions: If multiple instances of the same category exist, the model must generate unique referring expressions to distinguish them.

Stage 2: Target-specific multiple-choice question answer generation. We generate the question stem and the multiple-choice options in two sequential sub-steps to ensure logical consistency.

Stage 2-1: Question-answer pair generation. First, we feed the scene JSON from Stage 1 and a list of target object IDs into the MLLM. We use a modular prompting approach: a general instruction template (Fig. 10) is combined with specific **Task Modules** depending on the question category. The General Template enforces that the answer must be derived *exclusively* from the provided scene JSON to prevent hallucinations. The Task Modules contain specific logic for our six tasks: Reference (Fig. 11), Attribute (Fig. 12), Spatial (Fig. 13), Feedback (Fig. 14), Counting (Fig. 15), and Temporal (Fig. 16).

Stage 2-2: Negative choices generation. To convert the QA pairs into challenging Multiple-Choice Questions (MCQs), we prompt the model to generate distractors based on the specific question type, as detailed in Fig. 17. For binary questions, the model is restricted to "Yes/No" options. For open-ended questions, we instruct the model to generate four "Hard Negatives" using a prioritized strategy: (1) Visible Sources: Attributes or locations of *neighboring* objects in the scene (e.g., if the target is red, select the color of a nearby blue cup). (2) Plausible Fakes: Attributes that are semantically valid for the category but visually incorrect. (3) Logical Opposites: For spatial or temporal relations (e.g., "Left" vs. "Right").

Stage 3: Deictic question rephrasing. In the final stage, we prompt GPT-4o to transform the structured, robotic questions into natural, spoken egocentric queries (Fig. 18). The prompt acts as both a linguistic rephraser and a quality control. The model is instructed to replace target placeholders with deictic pronouns ("this", "that"), while preserving the names of reference objects (anchors) that are not being pointed at (e.g., "Is **this** closer than the red book?"). Here, we enforce that the rephrased question must be ambiguous if read without seeing the pointing gesture.

Stage 1: Dense Scene Information Extraction

```
### Role:
You are a highly perceptive visual analysis expert, acting as a "Dense Scene Fact Miner."

### Task:
Analyze the provided video frames and generate a comprehensive, structured description of **ALL salient
objects** visible in the scene.

### CONSTRAINT
 1. Hand-Agnostic: You MUST NOT mention "hand," "fingers," "arm," or "person" in the object descriptions. -
**BAD:** "held by a hand," "being moved by a user."
-   **GOOD:** "The object is moving upwards," "situated on the table."
 2. Discrimination: If multiple objects of the same category are present (e.g., three different bottles), you
MUST provide unique `referring_expressions` and `attributes` for each so they can be distinguished (e.g.,
"The green glass bottle" vs "The clear plastic bottle").

### OUTPUT FORMAT (Strict JSON List)
Return a single JSON list containing a dictionary for every significant object visible.
```json [ { "id": "object_01", "identity": "The common name (e.g., 'glass jar').", "object_category": "Broad
category (e.g., 'container').", "part_level": "Description if it is a part (e.g., 'lid'), otherwise null.",
"referring_expression": "A unique phrase distinguishing this object from others (e.g., 'the tall glass jar on
the left').", "attributes": { "color": "Specific color.", "material": "Apparent material.", "shape": "Primary
shape.", "contents": "Visible contents (e.g., 'pasta', 'empty').", "state": "Condition (e.g., 'open',
'closed', 'dirty')." }, "temporal_summary": "Concise summary of this specific object's movement or lack
thereof (e.g., 'Stationary throughout').", "function": "Typical purpose (e.g., 'storing food').",
"spatial_location": "Location relative to the scene (e.g., 'On the middle shelf, next to the red bowl')." },
{ "id": "object_02", ... } ]

### Instruction:
1. Scan the scene from left to right, foreground to background.
2. Identify every distinct, salient object.
3. Generate the JSON list strictly following the structure above.
```

Figure 9. **Prompt used for dense scene information extraction (Stage 1).** Instructions given to the MLLM to extract a structured JSON list of all salient objects.

Stage 2-1: Target-Specific Question Answer Pair Generation Template

```
### Role:
You are an expert region-level question answer generator for egocentric video datasets.

### Goal:
Generate 5 diverse and logically challenging Question-Answer pairs based on the provided scene data.

### Inputs:
• [TARGET_IDS]: An ordered list of object IDs to focus on (e.g., ["<obj_1>", "<obj_5>"]).
• [OBJECTS_DATA]: The JSON list of all visible objects (containing attributes, locations, and functions).


### Instruction:
1. Generate a total of 5 questions. using the placeholder <obj_ID>.
2. Always refer to the target object using its ID (e.g., `<obj_5>`).
3. The `correct_answer` must be derived *exclusively* from the provided JSON data. Do not hallucinate
   properties.
4. Do not stick to simple "What is X?" questions. Use comparisons, negations, and scenario-based phrasing
   where possible.

### Output format:
Return a single JSON list containing 5 objects.
```json
[{ "target_id": "<obj_X>","question": "The generated question string with placeholders","answer": "The exact
ground truth string"}, ... (total of 5)]```

<<TASK-SPECIFIC PROMPT & Examples>>

### TARGET_IDS: <<taret_object_ids>>
### OBJECTS_DATA: <<stage1_output>>
```

Figure 10. **General template for QA pair generation (Stage 2-1).** The base instruction template is used for all question types. It enforces that the ”correct answer” must be derived exclusively from the scene JSON provided in Stage 1 to prevent hallucinations.

Stage 2-1: Target-Specific Question Answer Pair Generation - Reference

```
### TASK DESCRIPTION: REFERENCE
**Goal:** Test the model's ability to identify the object's category or specific identity.
**Rules:**
1. **Identity:** Ask for the name/category (e.g., "What is <obj_X>?").
2. **differentiation:** If there are similar objects (e.g., two cups), ask a question that confirms specific
identity (e.g., "Is <obj_X> the red cup or the blue cup?").
3. **Negation:** Ask what the object is *not* (e.g., "Is <obj_X> a laptop?").

### EXAMPLE
**Input:**
TARGET_IDS:
[<obj_1>, <obj_2>]
OBJECTS_DATA:
[{"obj_id": "<obj_1>", "identity": "Espresso Machine", "object_category": "Appliance"},
{"obj_id": "<obj_2>": "Coffee Grinder", "object_category": "Appliance"}, ...]


**Output:**
[
  {
    "target_id": "<obj_1>",
    "question": "What specific appliance is <obj_1>?",
    "answer": "Espresso Machine"
  },
  {
    "target_id": "<obj_2>",
    "question": "What is <obj_2>?",
    "answer": "Coffee Grinder"
  }
]
```

Figure 11. **Task-specific prompt for Reference QA generation.** This task directs the model to generate questions that require identifying the specific description or category of the pointed-at object (e.g., “What is this?”).

Stage 2-1: Target-Specific Question Answer Pair Generation - Attribute

```
### TASK DESCRIPTION: ATTRIBUTE
**Goal:** Test the model's perception of fine-grained visual details.
**Rules:**
1. **Direct Query:** Ask for a specific attribute value from the JSON (Color, Shape, Material, State).
2. **Comparison:** Compare the target to another visible object (e.g., "Is <obj_X> the same material as
<obj_Y>?").
3. **State:** Ask about the condition (e.g., "Is <obj_X> currently open or closed?").

### EXAMPLE
**Input:**
TARGET_IDS:
["<obj_1>", "<obj_2>"]
OBJECTS_DATA:
[
  {"obj_id": "<obj_1>", "identity": "Red Apple", "attributes": {"color": "red", "state": "whole"}},
  {"obj_id": "<obj_2>", "identity": "Green Apple", "attributes": {"color": "green", "state": "sliced"}}
]

**Output Generation:**
[
  {
    "target_id": "<obj_1>",
    "question": "What is the color of <obj_1>?",
    "answer": "Red"
  },
  {
    "target_id": "<obj_1>",
    "question": "Is <obj_1> sliced like <obj_2>?",
    "answer": "No"
  }
]
```

Figure 12. **Task-specific prompt for Attribute QA generation.** This module focuses on fine-grained visual details, instructing the model to query properties such as color, shape, material, or state (e.g., “Is this sliced?”).

Stage 2-1: Target-Specific Question Answer Pair Generation - Spatial

```
### TASK DESCRIPTION: SPATIAL
**Goal:** Test the model's understanding of relative position and depth.
**Rules:**
1. **Relative Location:** Ask where the target is relative to a landmark object (e.g., left/right/above/
below).
2. **Proximity:** Ask about distance (e.g., "Is <obj_X> closer to the sink than <obj_Y>?").
3. **Constraint:** You MUST verify the relationship using the provided `spatial_location` or scene
description in the JSON.

### EXAMPLE
**Input:**
TARGET_IDS:
["<obj_1>", "<obj_2>"]
OBJECTS_DATA:
[
  {"obj_id": "<obj_1>", "identity": "Sponge", "spatial_location": "Inside the sink", "bbox":[x1,y1,x2,y2],},
  {"obj_id": "<obj_2>", "identity": "Soap dispenser", "spatial_location": "To the right of the sink", "bbox":
[x1,y1,x2,y2],}
  {"obj_id": "<obj_3>", "identity": "Sink", "spatial_location": "Next to the fridge", "bbox":[x1,y1,x2,y2],}
]

**Output Generation:**
[
  {
    "target_id": "<obj_1>",
    "question": "Where is <obj_1> located relative to the sink?",
    "answer": "Inside"
  },
  {
    "target_id": "<obj_2>",
    "question": "Is <obj_2> to the left of the sink?",
    "answer": "No"
  }
]
```

Figure 13. **Task-specific prompt for Spatial QA generation.** This task generates questions regarding the relative position or depth of the target object compared to other landmarks in the scene (e.g., “Is this to the left of the sink?”).

Stage 2-1: Target-Specific Question Answer Pair Generation - Feedback

```
### TASK DESCRIPTION: FEEDBACK
**Goal:** Test reasoning about user intent, affordance, and safety.
**Rules:**
1. **Goal Suitability:** Create a user goal (e.g., "I am thirsty") and ask if the target is suitable.
2. **Comparison:** Ask which object is better for a specific task.
3. **Safety:** Ask if an interaction is safe based on attributes (e.g., "Can I put <obj_X> (metal) in the
microwave?").

### EXAMPLE
**Input:**
TARGET_IDS:
["<obj_1>", "<obj_2>"]
OBJECTS_DATA:
[
   {"obj_id": "<obj_1>", "identity": "Glass Vase", "function": "Decoration", "attributes": {"material":
"Glass"}},
   {"obj_id": "<obj_2>", "identity": "Plastic Cup", "function": "Drinking", "attributes": {"material":
"Plastic"}}
]

**Output Generation:**
[
  {
    "target_id": "<obj_2>",
    "question": "I am thirsty. Should I use <obj_2> to drink water?",
    "answer": "Yes"
  },
  {
    "target_id": "<obj_1>",
    "question": "Is <obj_1> suitable for drinking water?",
    "answer": "No"
  }
]
```

Figure 14. **Task-specific prompt for Feedback QA generation.** This task tests reasoning about user intent and affordance, generating questions about whether a specific object is suitable for a stated goal (e.g., “I am thirsty. Can I drink this?”).

Stage 2-1: Target-Specific Question Answer Pair Generation - Counting

```
### TASK DESCRIPTION: COUNTING
**Goal:** Test visual search and enumeration of number of objects.
**Rules:**
1. Identify the category of the target (e.g., "Bottle"). Count ALL objects in `OBJECTS_DATA` that share this
category.
2. Ask "How many of **these** (<obj_X>) are there?"
3. Ensure the question specifies *what* to count (e.g., "How many red objects...", "How many bottles...").

### EXAMPLE
**Input:**
TARGET_IDS:
["<obj_1>", "<obj_3>"]
OBJECTS_DATA:
[
  {"obj_id": "<obj_1>", "object_category": "Pen", "attributes": {"color": "Blue"}},
  {"obj_id": "<obj_2>", "object_category": "Pen", "attributes": {"color": "Red"}},
  {"obj_id": "<obj_3>", "object_category": "Pencil", "attributes": {"color": "Yellow"}}
]

**Output Generation:**
[
  {
    "target_id": "<obj_1>",
    "question": "How many of <obj_1> are visible in the scene?",
    "answer": "2"
  },
  {
    "target_id": "<obj_3>",
    "question": "How many of <obj_3> are there?",
    "answer": "1"
  }
]
```

Figure 15. **Task-specific prompt for Counting QA generation.** This task tests visual search capabilities by asking the model to enumerate instances of the pointed-at object category visible in the scene.

Stage 2-1: Target-Specific Question Answer Pair Generation - Temporal

```
### TASK DESCRIPTION: TEMPORAL
**Goal:** Test the model's ability to recall the chronological sequence of interactions.
**Critical Assumption (The Timeline):** The `TARGET_IDS` list provided in the input acts as the **Interaction
Log**.
* **Index 0** = The **1st** object pointed at.
* **Index 1** = The **2nd** object pointed at.
* **Index N** = The **(N+1)th** object pointed at.
* *Note:* Ignore the numeric digits in the ID itself (e.g., `<obj_2>` might appear before `<obj_1>`).

**Rules:**
1. **Ordinality:** Ask "What is the [First/Second/Last] object pointed at?"
2. **Relative Sequence:** Ask "Did I point to <obj_X> before/after <obj_Y>?"
3. **Attribute Recall:** "What was the color of the object I pointed at *before* <obj_X>?"

### EXAMPLE
**Input:**
TARGET_IDS:
["<obj_2>", "<obj_1>"]
*(Meaning: First interaction was <obj_2>, Second interaction was <obj_1>)*

OBJECTS_DATA:
[
  {"obj_id": "<obj_1>", "identity": "Lamp", "attributes": {"color": "Silver"}},
  {"obj_id": "<obj_2>", "identity": "Book", "attributes": {"color": "Red"}}
]

**Output Generation:**
[
  {
    "target_id": "<obj_1>",
    "question": "What is the second object I pointed at?",
    "answer": "Lamp"
  },
  {
    "target_id": ["<obj_1>","<obj_2>"],
    "question": "Did I point at <obj_1> after <obj_2>?",
    "answer": "Yes"
  }
]
```

Figure 16. **Task-specific prompt for Temporal QA generation.** This task challenges the model's ability to recall the chronological sequence of pointing gestures (e.g., "What is the second object I pointed at?").

Stage 2-2: Negative Choices Generation

```
### Role:
You are an expert Multiple-Choice Question Generator specializing in "Hard Negative" mining.

### Task:
Analyze a pre-generated Question-Answer pair and generate high-quality distractors to complete the Multiple
Choice Question (MCQ).

### DISTRACTOR RULES
1. Case 1: Binary Questions (Yes/No)
* **Trigger:** If question starts with "Is", "Are", "Do", "Does", "Did", "Can", "Should".
* **Action:** Generate **only two options**: "(A) yes", "(B) no".

2. Case 2: Standard Questions (Open-ended)
 • Trigger: All other questions (What, Where, How many, Which).
 • Action: Generate **4 distractors** (Total 5 options including the answer).
 • Strategy:
     1.  **[Visible Source]:** (Highest Priority) Pick attributes/locations of **other visible objects** in
`[SCENE_CONTEXT]`.
    2.  **[Plausible Fake]:** Invent values that make sense for the category but are wrong.
    3.  **[Logical Opposite]:** For spatial (e.g., Left vs Right).

### OUTPUT FORMAT
Return the exact input JSON, but append `options` and `distractor_rationale`.

### EXAMPLE (Standard)
**Input:** `{"question": "What material is <obj_1>?", "answer": "Glass", "target_id": "<obj_1>"}OBJECTS_DATA`
**Output:**
```json
{
  "target_id": "<obj_1>",
  "question": "What material is <obj_1>?",
  "answer": "Glass",
  "options": ["(A) Wood", "(B) Plastic", "(C) Glass", "(D) Metal", "(E) Ceramic", "(F) Paper"],
  "distractor_rationale": {
    "Wood": "Source: Visible Source (Table). Rationale: Confusing target with nearby object."
  }
}
```
```

Figure 17. **Prompt used for negative choices generation (Stage 2-2).** Instructions for generating negative choices. The model is prompted to select attributes from neighboring objects or plausible but incorrect properties to create challenging multiple-choice options.

Stage 3: Deictic Question Rephrasing

```
### Role: You are a linguistic expert and Quality Control Auditor for an Egocentric Video dataset.

 • Input:
A JSON list of structured question objects (containing question, options, answer, and metadata).

### Task:
Transform the robotic question string (e.g., "What is <obj_1>?") into a natural, spoken question using
deictic pronouns ("this", "that").

### Criteria:
You must evaluate every rephrased question against these two rules. If a question cannot satisfy both,
DISCARD IT (do not include it in the output).
    i. Deictic Ambiguity (The "Blindfold" Test): The rephrased question MUST BE AMBIGUOUS if read without
       seeing the video.
     • CONSTRAINT: You MUST NOT reveal the identity of the target object in the question stem.
     • Bad (Too specific): "What color is this apple?" (FAIL: The user knows it's an apple without looking).
     • Bad (Too specific): "Is the mug to the left of the jar?" (FAIL: The user knows the target is a mug).
     • Good (Ambiguous): "What color is this?" (PASS: User must look at the pointing gesture).
     • Good (Ambiguous): "Is this to the left of the jar?" (PASS).
   ii. Correctness & Preservation: The rephrased question must strictly maintain the original logic so the
       answer remains correct.
     • CONSTRAINT: Do not change the question type. If the original asked for "Material", do not change it to
       "Texture".
     • CONSTRAINT: Do not alter the options or answer fields.

### REPHRASING INSTRUCTIONS
    i. Target Replacement: Replace the target object ID (e.g., <obj_1>) with "this", "that", "it", "these", or
       "the one".
   ii. Reference Preservation: If the question compares the target to a second object (that is NOT being
       pointed at), refer to the second object by its natural name (e.g., "the red book") to ensure the
       sentence is grammatically coherent.
        • Example: "Is <obj_1> closer than <obj_2>?" -> "Is this closer than the red book?"

### OUTPUT FORMAT
Return a JSON list of the valid, filtered objects.
If a question fails the criteria and cannot be fixed, exclude it from the list.
Now, rephrase the provided question answer pair.

### INPUT:
<<stage_2_output_json>>

### OUTPUT:
```

Figure 18. **Prompt used for deictic question rephrasing (Stage 3).** The final prompt in the pipeline, where GPT-4o acts as a linguistic expert to convert structured queries into natural, spoken questions. It replaces explicit object names with deictic pronouns ("this", "that") to ensure the question requires visual grounding to be answered.

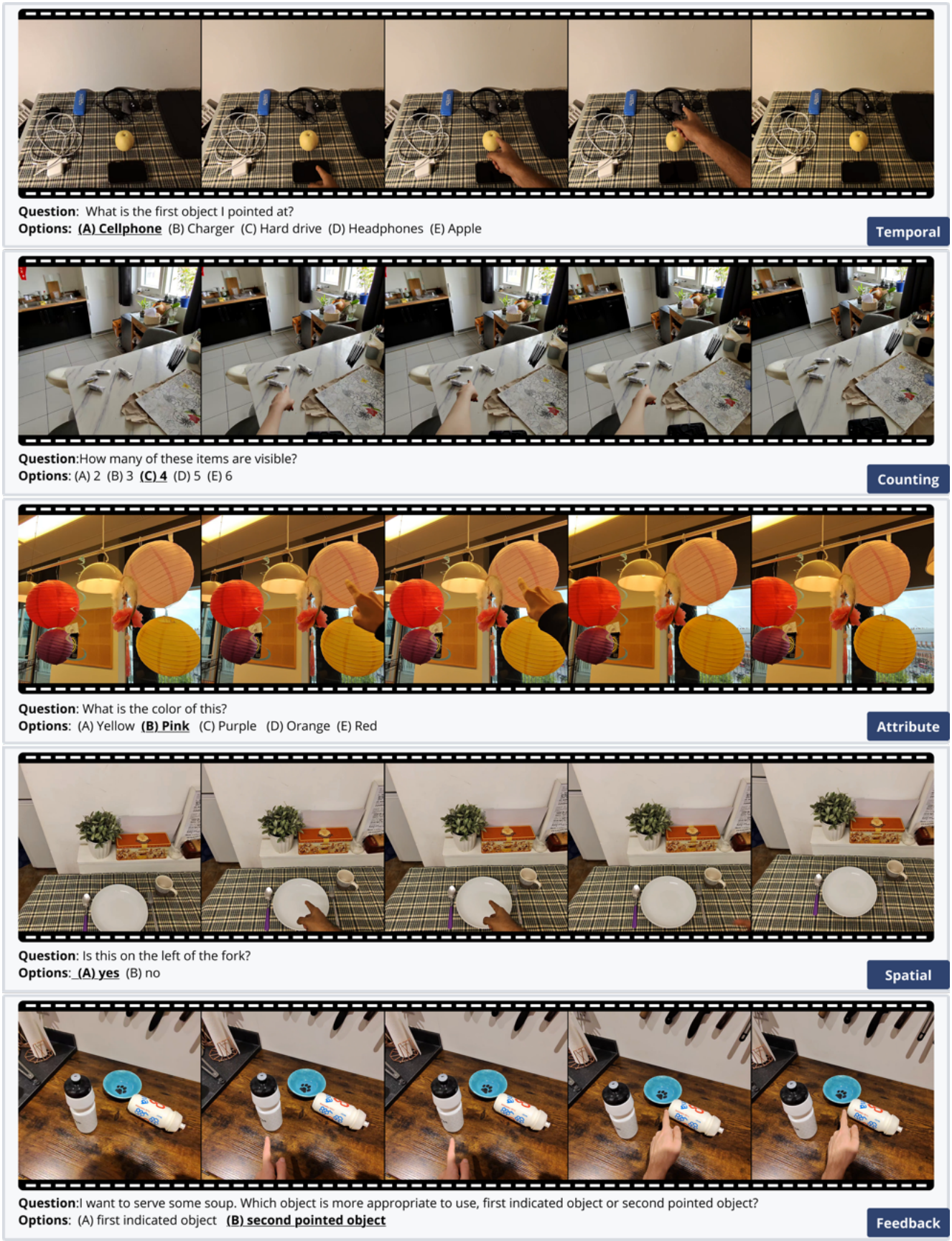

Figure 19. **Examples from real-world indoor scenes.** Samples from the real-world split of the dataset, captured in indoor environments. Each row displays the sample video frames alongside the corresponding question-answer pair generated by our pipeline. The correct answer is bolded and underlined within the options.

Figure 20. **Examples from real-world outdoor scenes.** Samples from the real-world split of the dataset, captured in outdoor environments. Each row displays the sample video frames alongside the corresponding question-answer pair generated by our pipeline. The correct answer is bolded and underlined within the options.

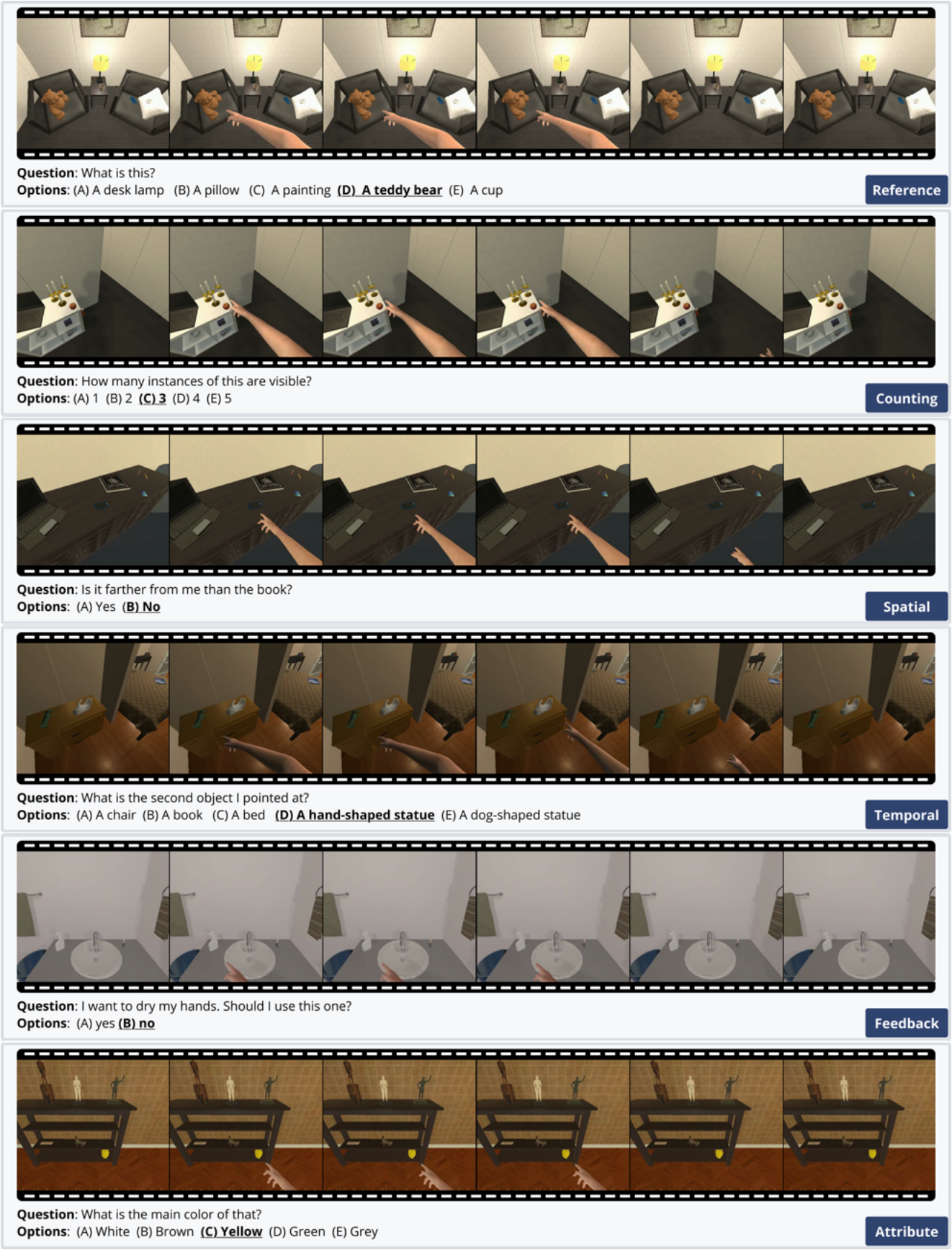

Figure 21. **Examples from synthetic training data.** Samples from the large-scale synthetic training set generated via AI2-THOR.