# TOWARDS A SYSTEMATIC STUDY OF EMBODIED SYSTEM 2 OVER FULL TASK EXECUTION CYCLES

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

## ABSTRACT

An embodied brain, or System 2, must turn an abstract goal into actions while discovering relevant scene conditions and checking whether executed skills achieved their intended effects. Open-loop evaluations cannot test observations shaped by an agent's decisions, while some closed-loop evaluations provide limited evidence during execution or assume idealized skill outcomes. We introduce SCOPER, a suite for studying observation-grounded planning, execution, and recovery. SCOPER-bench combines high-level goals, ongoing visual observations, a shared rule-based System 1 for low-level actions, and controlled execution failures. Its construction and validation pipeline yields 870 tasks across BEHAVIOR and ProcTHOR, complemented by about 3,500 situated questions tied to the evaluated agent's execution. Benchmark results reveal difficulties in retaining scene evidence, monitoring progress, and recovering from unexpected outcomes. To address these difficulties, SCOPER-harness combines Evidence Memory and Progress Verification with a dynamic task agenda and bounded exploration, without additional model training. Across two evaluated open-source backbones and two simulators, the harness improves task success by 11.1–28.6%; a harnessed 9B model outperforms a directly executed 27B model on both simulators. In an offline next-action evaluation using observations from 200 real-robot demonstrations, the harness also improves full-call accuracy by 14.7% and 15.3%. SCOPER provides a setting for diagnosing and improving embodied reasoning throughout a complete task execution cycle.

## 1 INTRODUCTION

An embodied agent must do more than turn a goal into a sequence of actions (Huang et al., 2022b; Singh et al., 2022; Song et al., 2022). Its high-level controller, which we call the embodied brain or System 2, must decide what information to seek, interpret visual observations (Driess et al., 2023; Zeng et al., 2022), select skills (Mu et al., 2023; Ahn et al., 2024), and determine whether those skills achieved their intended effects (Guo et al., 2023; Liu et al., 2023b). A low-level System 1 executes the selected skills, following the separation of language-based planning and skill execution explored in prior embodied systems (Ahn et al., 2022; Huang et al., 2022c; Liang et al., 2022; Huang et al., 2023). In a partially observed environment, exploration may reveal new requirements, while an unsuccessful action may invalidate the current plan (Guo et al., 2023; Liu et al., 2023b). These demands are coupled: what the agent observes shapes what it should do (Wu et al., 2023b;a), and what happens after it acts shapes what it should do next. Studying them together is essential to understanding whether an embodied brain can guide long-horizon behavior through a complete closed-loop task (Yenamandra et al., 2023).

Figure 1 summarizes these demands and their treatment by SCOPER-bench and SCOPER-harness. Existing evaluations inadequately capture these coupled demands. Open-loop evaluations based on recorded egocentric videos assess interpretation of fixed observations (Qiu et al., 2024; Dang et al., 2025; Majumdar et al., 2024; Chen et al., 2023; Jia et al., 2022; Cheng et al., 2024), but cannot test whether an agent acquires missing evidence, adjusts task decomposition through interaction, or recovers from its own execution failures. Existing closed-loop benchmarks (Yang et al., 2025b; Cheng et al., 2025; Srivastava et al., 2021) also leave gaps when they omit intermediate execution observations, assume idealized execution, or provide detailed procedural instructions. These choices limit evaluation of evidence accumulation, outcome monitoring, and recovery, and can reduce planning



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

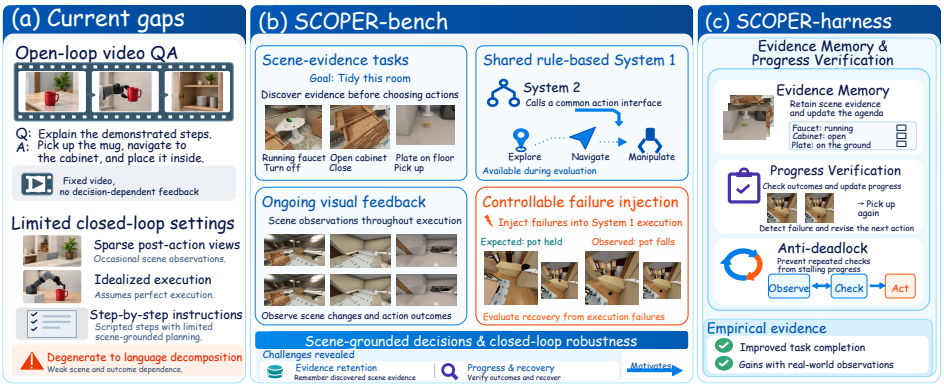


Figure 1: Embodied System 2 throughout closed-loop task execution. Open-loop reference trajectories cannot test decision-dependent interaction; closed-loop settings omitting execution-time observations or failures leave key capabilities untested. The SCOPER suite combines high-level goals, ongoing visual observations, and controlled failures in SCOPER-bench, with Evidence Memory and Progress Verification in SCOPER-harness.

to instruction-driven subtask sequencing with limited need for visual reasoning. The action–result loop alone therefore cannot establish conditions for studying embodied System 2.

We introduce the **SCOPER suite** for studying System 2 in closed-loop, observation-grounded planning, execution, and recovery, comprising **SCOPER-bench** and **SCOPER-harness**. SCOPER-bench requires agents to infer subtasks from abstract goals and multi-frame visual observations throughout execution; construction-time decompositions serve validation. A unified rule-based System 1 supplies shared execution infrastructure, and a scalable construction and validation pipeline yields 870 tasks across two simulators (Section 3). Controlled failure injection tests whether agents detect unmet expectations and replan to resume progress. Evaluations reveal persistent difficulties in acquiring and maintaining scene evidence for planning, monitoring outcomes, and recovering from failures.

SCOPER-harness addresses these difficulties through **Evidence Memory**, which maintains explicit language descriptions of observed scene information, and **Progress Verification**, which checks expected outcomes against observations after each execution step. A dynamic agenda organizes pending subtasks and is updated using accumulated evidence and verification results. Because repeated exploration and checking can delay execution, an anti-deadlock mechanism limits unproductive repetition to balance information gathering with task advancement.

Experiments show that complete execution remains challenging even with shared low-level skills: the strongest directly evaluated model achieves 50.5% task success on BEHAVIOR and 35.1% on ProcTHOR. Across evaluated open-source models, SCOPER-harness consistently improves performance, with gains of up to 28.6 percentage points, enabling a smaller model to perform comparably to the strongest evaluated model. Evaluated embodied-finetuned models also underperform general-purpose models, indicating that embodied specialization does not necessarily yield stronger closed-loop reasoning in this setting. Tests with real-world observations further show that benchmark-guided harness refinements improve performance, particularly in memory- dependent and failure-recovery cases.

Our contributions are threefold: (1) **SCOPER-bench:** A validated cross-simulator benchmark jointly testing observation-grounded task decomposition, execution, and failure recovery. (2) **SCOPER-harness:** A harness connecting scene evidence and verified outcomes to adaptive task agendas and sustained execution. (3) **Experiments and findings:** We identify gaps in scene evidence discovery and failure verification and design a targeted harness that consistently improves performance across evaluated models, with further gains on real-world observation tests.

## 2 Related Work

**Open-loop Evaluation of Embodied Reasoning.** Recent benchmarks evaluate embodied understanding and planning from recorded egocentric observations by asking the models to answer some

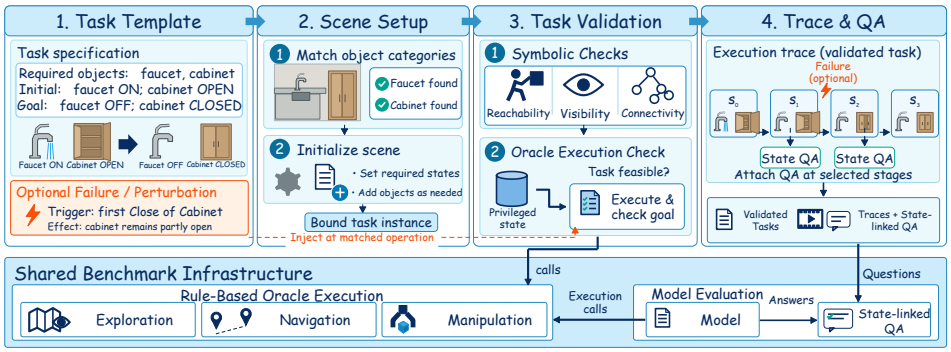


Figure 2: SCOPER-bench construction. Scene-grounded templates undergo static checks and privileged Oracle System 1 execution, including failures and recovery. Traces support situated QA construction; evaluation answers follow the evaluated episode's state. Average steps serve as a reference metric.

open-loop questions. (Grauman et al., 2021; 2023; Cheng et al., 2023; Mangalam et al., 2023; Yang et al., 2025a). EgoPlan-Bench2 (Qiu et al., 2024) tests planning in everyday scenarios, and ECBench (Dang et al., 2025) examines diverse embodied cognitive abilities. These protocols probe reasoning over visual evidence, but observations remain independent of the evaluated model's decisions. They therefore cannot establish whether an agent can acquire missing evidence through action, revise task decomposition as execution unfolds, or recover from its own execution failures. SCOPER-bench tests these demands through decision-dependent observations and outcome-guided planning and recovery.

**Closed-loop Embodied Benchmarks.** Interactive benchmarks (Shridhar et al., 2019; 2020; Padmakumar et al., 2021; Puig et al., 2018) increasingly evaluate embodied agents through simulated task execution (Savva et al., 2019; Szot et al., 2021; Puig et al., 2023; Deitke et al., 2020; Mees et al., 2021; Liu et al., 2023a). EmbodiedBench (Yang et al., 2025b) and EmbodiedEval (Cheng et al., 2025) extend evaluation to visual navigation, interaction, and planning, but their standard stepwise interfaces emphasize observations at action boundaries rather than evidence acquired within skill execution. Moreover, action-validity feedback does not systematically test whether agents detect and recover from deviations between intended and actual skill effects. BEHAVIOR (Srivastava et al., 2021; Li et al., 2024; 2021) evaluates complex household activities through integrated planning and control, making high-level reasoning difficult to disentangle from low-level execution in end-to-end performance. SCOPER-bench isolates these System 2 demands using shared, controllable System 1 skills, multi-frame execution observations, explicit failure injection, and high-level goals.

**Agentic designs for embodied intelligence.** Embodied frameworks combine planning, memory, and execution feedback to sustain long-horizon behavior (Rana et al., 2023; Wang et al., 2023a; Yao et al., 2022; Shinn et al., 2023; Wang et al., 2023b). Cortex (Peng et al., 2026) aligns VLM planning with VLA execution through standardized subtasks and semantic memory; Thea (Wang et al., 2026) integrates persistent scene context with action-outcome evaluation; Mimir (Xu et al., 2026) grounds an evolving task agenda in world memory and explores under insufficient evidence. SCOPER-harness emphasizes how scene discovery determines an abstract goal's required work, coupling exploration, evidence-driven agenda revision, and verified progress (Section 4).

## 3 SCOPER-BENCH

We test System 2 information acquisition, goal decomposition, subtask selection, and plan revision from observations and outcomes. Instructions specify requirements, not action sequences; completion and situated question answering (QA) assess execution and scene/task understanding.

### 3.1 TASK FORMULATION AND EVALUATION PROTOCOL

**Task groups.** We organize the benchmark into four groups. **Routine execution** specifies concrete task requirements while leaving action selection and sequencing to the agent. **Observation-dependent** covers state-dependent execution, scene inspection, and responses to task updates. **Fail-**

**ure recovery** covers initially present anomalies, controlled execution failures, and mid-execution disturbances that require corrective actions or replanning. **Composite** combines multiple subtasks, requiring agents to maintain progress and reason over cross-step dependencies; these subtasks can draw from the other three groups.

**Observation-dependent decisions.** Tasks differ in their information needs before acting. For example, "put the cup in the cabinet" identifies the object and destination, whereas "inspect the room, clean the dishware that needs cleaning, and put it away" requires discovering which objects need attention. Observation remains necessary after acting: following a placement request, the agent must inspect whether the object reached its destination and assess further action needs.

**Task representation.** A task instance is represented as $\mathcal{T} = (x, \mathcal{R}, \mathcal{C}_0, \mathcal{G}, \mathcal{E}, \mathcal{Q})$, where $x$ is the instruction, $\mathcal{R}$ specifies object-role constraints, $\mathcal{C}_0$ defines initial conditions, $\mathcal{G}$ contains goal predicates, $\mathcal{E}$ configures optional execution events, and $\mathcal{Q}$ contains attached questions. This configuration supports construction and evaluation. Evaluated System 2 uses instructions, visual observations, and prescribed public tool information; construction-time role bindings, reference decompositions, and privileged simulator state provide no planning guidance.

**Agent inputs and evaluation protocol.** System 2 receives instructions, tool descriptions, permitted vocabulary, decision history, and tool-returned images. Exploration and navigation return eight uniformly sampled execution frames; state-changing actions return before-and-after frames. Direct execution retains up to 32 frames; harness context routing appears in Section 4.1 and Appendix J. Execution ends on `stop` or step-budget exhaustion. Success requires both `stop` and the resulting environment state satisfying the template's goal predicates. Triggered questions use dedicated answer-only turns with the same context available to policy decisions. Observation settings, budgets, and scoring rules appear in Appendix A; QA activation and answering details appear in Appendix I.

### 3.2 Oracle System 1 and Controlled Execution

**Unified skill infrastructure.** Oracle System 1 provides rule-based active observation, navigation, and manipulation skills shared by evaluated models and the construction-time teacher. Standardizing execution controls low-level variation without training a separate policy for each model. Complete skill definitions and object binding are given in Appendices E and H. Navigation uses A* search (Hart et al., 1968) over traversability maps derived from scene collision geometry. Manipulation bypasses low-level physical execution, directly updating object poses, holding or containment relations, and task-relevant states with synchronized visual representations. Navigation, scanning, and target search collect multi-frame observations along paths and viewpoints; state-changing actions expose their visual outcomes for subsequent System 2 decisions.

**Controlled failures and perturbations.** Template-defined events alter System 1 behavior or outcomes at designated stages, requiring inspection of deviations, identification of unmet requirements, and recovery or replanning. This combines shared execution infrastructure and controllable imperfect outcomes; backend implementations and failure mechanisms appear in Appendices E and F.

### 3.3 Task Construction and Validation

**Template-to-scene construction.** Figure 2 summarizes the shared construction pipeline in BEHAVIOR (Srivastava et al., 2021) and ProcTHOR (Deitke et al., 2022; Kolve et al., 2017), which diversify scene styles and layouts. Using template constraints, the constructor matches scenes and binds roles to objects, checking required objects and fixtures against interaction and spatial constraints, using scene profiles in ProcTHOR and scene–template matching with asset materialization in BEHAVIOR. Initialization sets object placements, containment, agent pose, and task-relevant states with synchronized visual representations. For dishware inspection, a dirty item is placed where exploration is needed to discover it; the instruction specifies inspection and handling requirements without revealing the target. Rule-based subtask decomposition guides execution validation. Configuration details appear in Appendix G.

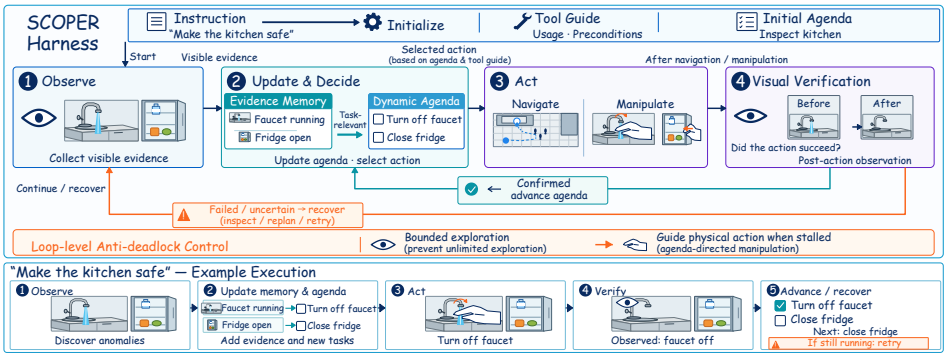


Figure 3: SCOPER-harness. Evidence Memory: agenda revision, action selection; Progress Verification: execution outcomes for planning, recovery; anti-deadlock control: balancing exploration and task advancement.

**Static and execution validation.** Static checks test role and condition consistency, initial-state reconstruction, nontrivial goals, and reachability. A privileged teacher follows the reference decomposition through Oracle System 1, checking goal satisfaction and configured events; failure-injection instances must recover to completion with failures enabled. Traces and videos support inspection. We construct 870 executable tasks from 35 base templates and their compositions (Appendix G).

**Situated QA attachment.** About 3,500 questions from eight QA templates probe object states, spatial relations, action outcomes, anomalies, new requirements, and task progress. Candidates derived from task semantics are screened against teacher traces for valid activation and resolvable answers. Evaluation triggers questions at configured actions, steps, or subtask milestones, with answers determined from the evaluated episode's corresponding state (Appendix I).

## 4 SCOPER-HARNESS

Prior embodied systems use environmental feedback and explicit memory to support planning (Huang et al., 2022c; Wang et al., 2026; Xu et al., 2026; Huang et al., 2022a; Jatavallabhula et al., 2023; Gu et al., 2023). Our evaluations expose two difficulties in closed-loop task execution: maintaining scene evidence to guide task decomposition and subsequent decisions, and detecting unsuccessful execution to support recovery. We introduce **SCOPER-harness**, a training-free harness that addresses these difficulties through **Evidence Memory** and **Progress Verification**. A dynamic agenda connects accumulated evidence to task requirements, while anti-deadlock control balances information gathering with task advancement. All model-based components share the same fixed-parameter backbone through role-specific prompts and use no additional privileged simulator information. Figure 3 summarizes the architecture.

### 4.1 EVIDENCE MEMORY AND ADAPTIVE TASK DECOMPOSITION

**Structured evidence memory.** Partial observations require retaining evidence without treating missing observations as state changes. We maintain $M_t = (F_t, I_t, O_t)$: facts store object properties, relations, and scene descriptions with evidence and validity annotations; incidents record execution discrepancies and handling status; obligations track outstanding observation or action needs linked to relevant objects and incidents.

**Instruction-derived agenda.** The agenda $A_t$ organizes required outcomes, ongoing constraints, and conditional requirements, while memory obligations track observation or handling needs. Initialization uses only the instruction, retaining unresolved conditions as inspection needs without assuming unseen states or fixing an action sequence. Items track status and supporting evidence.

**Observation-driven memory updates.** Before decisions, discovery extracts task-relevant objects and states from current and available execution-process images; reconciliation checks facts against new evidence. Confirmation refreshes evidence, contradiction invalidates facts, and newer scene

descriptions can supersede older ones; invisibility alone does not negate a fact. A deterministic commit layer checks references, evidence links, and consistency before committing model updates.

**Evidence-grounded agenda revision.** Committed memory drives agenda revision: the model combines the instruction, existing items, and new evidence to activate conditional requirements, add necessary work within task scope, merge duplicates, and update or reopen affected items. Inspection items become supported when the required information is obtained; execution items require verified effects, while ongoing constraints remain monitored. Unresolved inspection needs guide observation actions toward relevant objects or conditions. For example, "inspect the kitchen and turn off any running faucets" initializes inspection and conditional shutoff requirements. Observing running water supports the corresponding state check and activates shutoff work, but does not establish that the faucet has been turned off.

**Context Routing and Action Selection.** Deterministic routing selects committed memory within a bounded context, prioritizing relevant facts and unresolved needs while retaining evidence references and validity annotations. The policy receives the routed memory $\widetilde{M}_t$ together with the instruction $x$, recent images $V_t$, public action-request history $H_t$, agenda $A_t$, and control context $c_t$: $a_t = \pi_\theta(x, V_t, H_t, A_t, \widetilde{M}_t, c_t)$. The control context specifies available tools and arguments, the recovery phase, and action restrictions. The policy outputs one tool call with arguments. Request history records issued tool requests but provides no ground-truth success labels.

### 4.2 Progress Verification and Recovery

**Visual verification and recovery feedback.** Issuing an action does not establish progress. Navigation and manipulation therefore receive outcome verification; exploration supplies memory observations without separate verification. Given the tool request, expected state and visual effects, and immediate before-and-after images, the verifier returns success, failure, or ambiguity with visual evidence. Success commits supported effects and resolves satisfied needs; failure records discrepancies and recovery obligations; ambiguity leaves outcomes unconfirmed. Memory adopts the verifier's effect assessment without readjudicating the same images; later observations may revise it. In the faucet example, stopped flow supports completion; continued flow leaves shutoff unresolved and triggers recovery. The original request, evidence, and recovery state guide further observation, renewed approach, or retry. Verified subsequent effects update memory and the agenda, returning local recovery to task planning.

### 4.3 Anti-Deadlock Control

**Balancing exploration, execution, and recovery budgets.** Evidence maintenance and verification can encourage excessive exploration or conservative checking. The controller tracks scans, searches, and navigation by room/target, and observations and retries per incident. At exploration limits, it restricts tools and requests reselection to promote approach and interaction. The agenda guides planning; recovery and anti-deadlock rules constrain execution. Reselection and recovery are budgeted. Exhaustion terminates execution or permits continuation with unresolved state, depending on the recovery branch. Temporary planning assumptions remain separate from verified facts, cannot support agenda completion, and remain revisable by later evidence. Appendix J details memory and agenda records and states, routing order and capacities, recovery fields and transitions, counter updates and resets, branch rules, prompt responsibilities, and execution pseudocode.

## 5 Experiments

### 5.1 Experimental Setup

We evaluate GPT-6 Astra (OpenAI, 2026), Gemini-3.8-flash (Google DeepMind, 2026), Embodied-R1.5 (Yuan et al., 2026), RoboBrain2.5-8B (Tan et al., 2026), RynnBrain11-9B (Li et al., 2026), Molmo2-ER-4B (Fang et al., 2026), Qwen3.5-9B (Qwen Team, 2026a), and Qwen3.8-27B (Qwen Team, 2026b) in ProcTHOR (Deitke et al., 2022) and BEHAVIOR (Srivastava et al., 2021). All eight models follow the shared Oracle System 1 interface and input protocol (Section 3.1), using

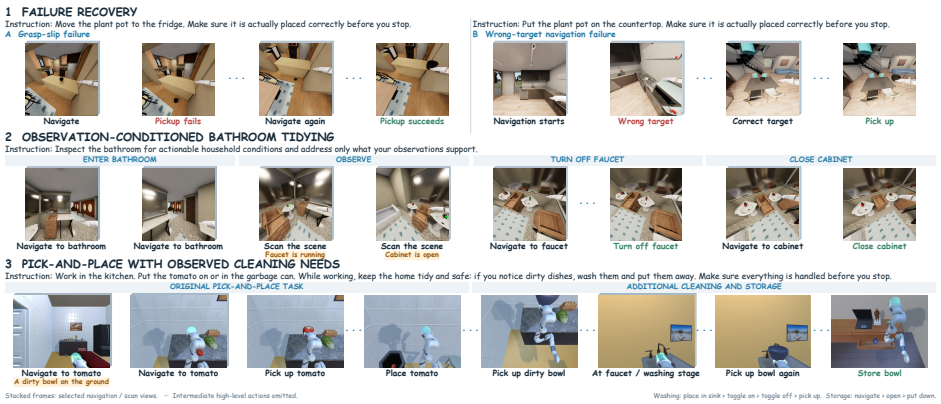


Figure 4: Example task traces: (1) recovery from controlled grasp and navigation failures; (2) observation-conditioned household actions identifying targets undisclosed in instructions; and (3) observation activating and fulfilling an instruction-specified cleaning requirement during placement.

$512 \times 512$ RGB observations without privileged simulator state. For Qwen3.5-9B and Qwen3.8-27B, we compare direct execution against the same-backbone, training-free SCOPER-harness (Section 4). Given the limited gains and substantially higher cost of pairing GPT-6-Astra with the harness, we exclude this configuration from the full comparison. Across methods, atomic/two-subtask/three-subtask episodes allow 24/36/48 high-level calls, including QA answers. Tool interfaces, observation processing, and model/harness configurations appear in Appendices A, E, and J; model-call overhead is described in Appendix L.

SR is the task success fraction; Prog./Cov. are task-averaged fractions of satisfied goal predicates/covered required actions. QA (T/A) denotes triggered/configured questions and correctly answered/triggered questions, respectively, counting triggered but unanswered questions as incorrect. Aggregates weight task metrics by task counts and QA metrics by their respective question denominators. Success conditions and detailed scoring rules appear in Appendix A. Additional action-step and blind evaluations appear in Appendices B and C. Figure 4 illustrates why fixed instruction-derived action sequences can be insufficient. Further examples appear in Appendix K.

### 5.2 Main Results

**General-purpose capability matters.** Table 1 shows that GPT-6 Astra leads direct execution with overall SRs of 50.5% on BEHAVIOR and 35.1% on ProcTHOR, versus 14.8% and 21.1% for Qwen3.8-27B. Its necessary-action coverage is higher on both backends; only one of its 500 failed episodes ended in a parsing error. These results suggest strengths in reliable tool use, multi-step progression, observation-grounded action selection, outcome monitoring, and recovery. Embodied-specialized models show no consistent advantage, suggesting that gains on open-loop embodied evaluations need not transfer to closed-loop execution. Nevertheless, 58.6% of Astra's failures exhausted the action budget, indicating limits in exploration efficiency and recovery planning.

**Harness gains across models and backends.** SCOPER-harness raises overall SR by 24.8/28.6 percentage points on BEHAVIOR and 16.9/11.1 on ProcTHOR for Qwen3.5-9B/Qwen3.8-27B, respectively. SR, progress, and necessary-action coverage improve in every task group for both models on both backends. Qwen3.5-9B with SCOPER-harness surpasses direct Qwen3.8-27B execution on both backends. Additional inference calls organizing Evidence Memory, adaptive planning, and recovery thus improve fixed-model performance.

**Situated QA complements execution metrics.** GPT-6 Astra leads overall QA trigger rates on both backends while maintaining high accuracy on triggered questions, consistent with its task performance. SCOPER-harness increases trigger rates in all four backbone–backend combinations and triggered accuracy in three. Accuracy must be read alongside trigger rate: execution paths activate different question sets, precluding an identical-question interpretation.

Table 1: BEHAVIOR and ProcTHOR results by task group (%). QA reports trigger rate / triggered answer accuracy; Δ denotes the change from the corresponding direct-execution baseline in percentage points. Bold indicates the best result among non-API models within each backend.

| Model | Routine Execution | | | | Observation-dependent | | | | Failure Recovery | | | | Composite | | | | Overall | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SR | Prog. | Cov. | QA (T / A) | SR | Prog. | Cov. | QA (T / A) | SR | Prog. | Cov. | QA (T / A) | SR | Prog. | Cov. | QA (T / A) | SR | Prog. | Cov. | QA (T / A) |
| **BEHAVIOR** | | | | | | | | | | | | | | | | | | | | |
| *API model* | | | | | | | | | | | | | | | | | | | | |
| GPT-6 Astra | 42.0 | 74.6 | 72.6 | 81.8 / 54.3 | 40.0 | 56.9 | 69.1 | 85.9 / 82.5 | 76.2 | 85.1 | 88.2 | 84.4 / 74.7 | 35.0 | 75.4 | 59.4 | 43.9 / 58.1 | 50.5 | 74.2 | 73.5 | 74.4 / 68.9 |
| Gemini-3.8-flash | 25.0 | 44.6 | 42.3 | 48.5 / 30.6 | 25.6 | 35.4 | 43.5 | 54.4 / 53.2 | 50.8 | 57.3 | 59.2 | 57.0 / 51.1 | 20.0 | 41.2 | 32.4 | 24.2 / 31.9 | 31.9 | 45.7 | 45.4 | 46.5 / 44.1 |
| *Embodied models* | | | | | | | | | | | | | | | | | | | | |
| Embodied-R1.5 | 5.0 | 35.4 | 29.1 | 48.2 / **70.6** | 15.6 | 26.8 | 44.3 | 52.9 / 71.5 | 16.9 | 25.2 | 38.4 | 63.7 / 51.4 | 2.0 | 27.5 | 18.5 | 18.5 / 43.6 | 10.2 | 28.5 | 32.7 | 46.9 / 60.1 |
| RoboBrain2.5-8B | 1.0 | 27.0 | 16.2 | 28.0 / 51.8 | 11.1 | 13.7 | 29.3 | 29.3 / 72.4 | 10.8 | 18.9 | 27.7 | 43.3 / 44.6 | 1.0 | 24.3 | 12.4 | 11.1 / 60.6 | 6.2 | 21.0 | 21.7 | 29.0 / 53.6 |
| RynnBrain11-9B | 2.0 | 30.3 | 21.3 | 45.1 / 57.5 | 10.0 | 12.0 | 22.7 | 44.0 / 75.4 | 6.9 | 17.3 | 30.7 | 51.8 / 38.3 | 0.0 | 24.6 | 9.6 | 15.8 / 46.8 | 4.8 | 21.0 | 21.7 | 39.9 / 52.8 |
| Molmo2-ER-4B | 0.0 | 28.5 | 13.1 | 40.4 / 46.7 | 3.3 | 5.4 | 11.5 | 41.3 / 67.3 | 13.1 | 21.9 | 18.1 | 50.3 / 33.3 | 0.0 | 22.0 | 5.5 | 12.8 / 29.0 | 4.8 | 20.0 | 12.5 | 37.0 / 44.4 |
| *Open-source models* | | | | | | | | | | | | | | | | | | | | |
| Qwen3.5-9B | 5.0 | 35.7 | 25.6 | 47.1 / 60.7 | 15.6 | 19.8 | 33.0 | 46.7 / 66.1 | 15.4 | 24.9 | 35.8 | 49.5 / 43.8 | 0.0 | 25.0 | 13.2 | 15.4 / 56.5 | 9.3 | 26.4 | 27.4 | 40.2 / 55.1 |
| Qwen3.5-9B + harness | 28.0 | 58.9 | 54.0 | 68.5 / 56.7 | 30.0 | 41.0 | 53.6 | 68.5 / 74.3 | 50.8 | 60.9 | 67.1 | 71.0 / 65.0 | 22.0 | 56.1 | 41.7 | 33.0 / 59.6 | 34.1 | 55.0 | 55.0 | 60.8 / 64.2 |
| Δ vs. base (pp) | +23.0 | +23.3 | +28.5 | +21.4 / -4.1 | +14.4 | +21.2 | +20.6 | +21.8 / +8.2 | +35.4 | +36.0 | +31.3 | +21.5 / +21.2 | +22.0 | +31.1 | +28.5 | +17.6 / +3.1 | +24.8 | +28.6 | +27.6 | +20.6 / +9.1 |
| Qwen3.8-27B | 13.0 | 44.5 | 43.2 | 60.9 / 49.2 | 14.4 | 28.7 | 46.7 | 65.6 / 62.4 | 20.8 | 39.4 | 55.1 | 75.8 / 50.0 | 9.0 | 46.2 | 35.8 | 42.3 / **82.5** | 14.8 | 39.9 | 45.9 | 62.1 / 57.9 |
| Qwen3.8-27B + harness | **36.0** | **68.7** | **66.8** | **78.0** / 52.4 | **34.4** | **50.6** | **64.1** | **80.9 / 78.1** | **65.4** | **75.8** | **81.2** | **83.0 / 69.6** | **30.0** | **70.3** | **55.3** | **43.8** / 63.5 | **43.3** | **67.4** | **67.9** | **72.0 / 66.2** |
| Δ vs. base (pp) | +23.0 | +24.2 | +23.5 | +17.1 / +3.2 | +20.0 | +21.9 | +17.4 | +15.2 / +15.7 | +44.6 | +36.5 | +26.0 | +7.2 / +19.6 | +21.0 | +24.1 | +19.5 | +1.6 / -19.1 | +28.6 | +27.5 | +22.0 | +9.9 / +8.4 |
| **ProcTHOR** | | | | | | | | | | | | | | | | | | | | |
| *API model* | | | | | | | | | | | | | | | | | | | | |
| GPT-6 Astra | 35.4 | 75.7 | 68.8 | 79.0 / 60.5 | 41.1 | 66.4 | 74.1 | 82.0 / 68.4 | 43.9 | 69.6 | 78.8 | 80.9 / 59.7 | 18.0 | 67.3 | 50.8 | 52.3 / 65.6 | 35.1 | 70.2 | 68.7 | 74.2 / 62.7 |
| Gemini-3.8-flash | 20.8 | 47.1 | 42.4 | 49.1 / 37.4 | 25.6 | 43.8 | 49.6 | 55.3 / 45.4 | 27.7 | 43.5 | 49.0 | 49.3 / 39.7 | 14.0 | 49.7 | 37.5 | 38.8 / 48.2 | 22.2 | 46.0 | 44.7 | 48.1 / 41.8 |
| *Embodied models* | | | | | | | | | | | | | | | | | | | | |
| Embodied-R1.5 | 3.9 | 34.1 | 15.7 | 36.0 / 63.5 | 15.6 | 18.3 | 32.9 | 42.0 / 69.2 | 7.7 | 21.7 | 30.4 | 59.0 / **58.9** | 1.0 | 29.1 | 12.8 | 19.4 / 30.4 | 6.7 | 26.2 | 22.7 | 40.2 / 59.1 |
| RoboBrain2.5-8B | 1.5 | 26.9 | 11.8 | 21.5 / 46.3 | 10.0 | 14.4 | 21.7 | 25.9 / 60.6 | 9.2 | 22.7 | 24.8 | 40.7 / 44.2 | 3.0 | 28.7 | 16.2 | 19.4 / 55.4 | 5.8 | 23.6 | 18.5 | 27.5 / 49.4 |
| RynnBrain11-9B | 2.3 | 33.1 | 16.7 | 35.2 / 51.5 | 7.8 | 12.8 | 18.5 | 32.6 / 60.2 | 2.3 | 24.6 | 20.9 | 35.5 / 38.2 | 0.0 | 26.4 | 13.2 | 18.3 / 52.8 | 2.9 | 25.1 | 17.5 | 31.0 / 49.0 |
| Molmo2-ER-4B | 0.0 | 28.7 | 7.3 | 16.3 / 58.1 | 1.1 | 5.6 | 4.8 | 12.9 / 51.5 | 0.8 | 14.6 | 5.8 | 23.5 / 28.9 | 0.0 | 18.9 | 2.5 | 5.5 / 25.0 | 0.4 | 17.8 | 5.3 | 15.4 / 41.3 |
| *Open-source models* | | | | | | | | | | | | | | | | | | | | |
| Qwen3.5-9B | 0.8 | 34.1 | 25.8 | 44.1 / 55.4 | 18.9 | 25.6 | 40.9 | 50.6 / 62.8 | 8.5 | 29.2 | 43.6 | 64.5 / 44.9 | 2.0 | 35.8 | 23.5 | 31.1 / 75.6 | 6.9 | 31.4 | 33.4 | 48.5 / 55.7 |
| Qwen3.5-9B + harness | 20.8 | 59.1 | 51.9 | 65.6 / 58.2 | 32.2 | 49.6 | 60.8 | 69.5 / 66.1 | 30.0 | 53.5 | 64.7 | 74.3 / 54.5 | 12.0 | 54.6 | 39.7 | 43.7 / 69.8 | 23.8 | 54.6 | 54.7 | 64.1 / 60.3 |
| Δ vs. base (pp) | +20.0 | +25.0 | +26.1 | +21.5 / +2.8 | +13.3 | +24.0 | +19.9 | +18.9 / +3.3 | +21.5 | +24.2 | +21.1 | +9.8 / +9.5 | +10.0 | +18.8 | +16.2 | +12.5 / -5.7 | +16.9 | +23.2 | +21.2 | +15.6 / +4.7 |
| Qwen3.8-27B | 16.2 | 50.2 | 46.5 | 65.6 / **64.8** | 30.0 | 44.2 | 54.0 | 71.0 / **79.0** | 26.2 | 46.7 | 57.4 | 72.3 / 52.7 | 13.0 | 57.4 | 48.1 | **55.7 / 83.9** | 21.1 | 49.6 | 51.5 | 66.4 / **67.4** |
| Qwen3.8-27B + harness | **33.1** | **71.1** | **64.6** | **76.5** / 61.7 | **37.8** | **61.5** | **70.0** | **79.8** / 70.4 | **39.2** | **64.7** | **74.4** | **79.1** / 58.5 | **17.0** | **65.3** | **50.3** | 53.0 / 69.6 | **32.2** | **66.1** | **65.4** | **72.7** / 63.8 |
| Δ vs. base (pp) | +16.9 | +21.0 | +18.1 | +10.9 / -3.1 | +7.8 | +17.3 | +16.0 | +8.8 / -8.6 | +13.1 | +18.0 | +17.0 | +6.8 / +5.8 | +4.0 | +8.0 | +2.2 | -2.7 / -14.2 | +11.1 | +16.5 | +13.8 | +6.3 / -3.6 |

Table 2: Qwen3.8-27B harness component ablation results on ProcTHOR and BEHAVIOR

| Variant | Routine Execution | | | Observation-dependent | | | Failure Recovery | | | Composite | | | Overall | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SR | Prog. | Cov. | SR | Prog. | Cov. | SR | Prog. | Cov. | SR | Prog. | Cov. | SR | Prog. | Cov. |
| *ProcTHOR (450 episodes per variant)* | | | | | | | | | | | | | | | |
| Full harness | 33.1 | 71.1 | 64.6 | 37.8 | 61.5 | 70.0 | 39.2 | 64.7 | 74.4 | 17.0 | 65.3 | 50.3 | 32.2 | 66.1 | 65.4 |
| w/o Anti-deadlock | 21.5 | 45.5 | 40.5 | 24.4 | 41.7 | 47.5 | 23.9 | 42.5 | 48.0 | 11.0 | 40.7 | 30.4 | 20.4 | 42.8 | 41.8 |
| Δ vs. full (pp) | -11.5 | -25.7 | -24.2 | -13.3 | -19.8 | -22.5 | -15.4 | -22.2 | -26.4 | -6.0 | -24.7 | -19.9 | -11.8 | -23.3 | -23.5 |
| w/o Recovery | 34.6 | 74.0 | 67.1 | 42.2 | 61.1 | 70.8 | 27.7 | 50.7 | 62.4 | 18.0 | 60.3 | 49.3 | 30.4 | 61.7 | 62.5 |
| Δ vs. full (pp) | +1.5 | +2.9 | +2.5 | +4.4 | -0.3 | +0.8 | -11.5 | -14.0 | -12.0 | +1.0 | -5.1 | -1.1 | -1.8 | -4.4 | -2.8 |
| w/o Memory | 31.5 | 70.0 | 63.7 | 33.3 | 47.3 | 56.1 | 40.0 | 65.2 | 75.7 | 16.0 | 62.7 | 50.1 | 30.9 | 62.5 | 62.6 |
| Δ vs. full (pp) | -1.5 | -1.1 | -0.9 | -4.5 | -14.2 | -14.0 | +0.8 | +0.5 | +1.3 | -1.0 | -2.7 | -0.3 | -1.3 | -3.6 | -2.7 |
| *BEHAVIOR (420 episodes per variant)* | | | | | | | | | | | | | | | |
| Full harness | 36.0 | 68.7 | 66.8 | 34.4 | 50.6 | 64.1 | 65.4 | 75.8 | 81.2 | 30.0 | 70.3 | 55.3 | 43.3 | 67.4 | 67.9 |
| w/o Anti-deadlock | 22.0 | 41.1 | 39.4 | 20.0 | 31.2 | 38.1 | 38.5 | 47.5 | 50.1 | 18.0 | 43.0 | 32.4 | 25.7 | 41.4 | 40.8 |
| Δ vs. full (pp) | -14.0 | -27.5 | -27.4 | -14.4 | -19.4 | -26.0 | -26.9 | -28.4 | -31.1 | -12.0 | -27.4 | -22.9 | -17.6 | -26.0 | -27.2 |
| w/o Recovery | 38.0 | 71.3 | 67.5 | 34.4 | 51.0 | 63.7 | 21.5 | 40.9 | 58.1 | 29.0 | 68.3 | 53.6 | 30.0 | 56.8 | 60.5 |
| Δ vs. full (pp) | +2.0 | +2.6 | +0.7 | +0.0 | +0.4 | -0.4 | -43.8 | -35.0 | -23.1 | -1.0 | -2.1 | -1.6 | -13.3 | -10.6 | -7.5 |
| w/o Memory | 36.0 | 67.7 | 65.4 | 17.8 | 33.0 | 49.8 | 64.6 | 74.8 | 82.0 | 23.0 | 63.0 | 49.4 | 37.9 | 61.3 | 63.4 |
| Δ vs. full (pp) | +0.0 | -1.0 | -1.4 | -16.7 | -17.6 | -14.3 | -0.8 | -1.1 | +0.8 | -7.0 | -7.3 | -5.9 | -5.5 | -6.1 | -4.5 |

**Failure patterns motivate evidence and progress control.** Across the six open-source models' 4,813 failed normal-image episodes, 47.7% stopped prematurely and 30.7% exhausted the action budget, reflecting difficulties in completion assessment, state tracking, and sustained progress. At least three consecutive identical tool calls occurred in 40.2%, suggesting difficulty adapting unsuccessful strategies; parsing failures accounted for 12.8%, and reference-binding issues appeared in 17.3%. This motivates evidence-based completion checks, state tracking, repetition-aware recovery, and clearer tool/reference conventions. These indicators overlap rather than partition failure causes.

### 5.3 Harness Component Ablations

Table 2 separately ablates anti-deadlock control, explicit recovery, and Evidence Memory using Qwen3.8-27B on both backends. Each *w/o* disables only the named component; Recovery denotes Progress Verification and its associated recovery mechanism.

**Anti-deadlock control sustains progress.** Removing this component lowers overall SR by 11.8/17.6 percentage points on ProcTHOR/BEHAVIOR, reducing progress and necessary-action coverage in every task group. Trajectories show prolonged exploration and repeated confirmation

Table 3: Evaluation of normal execution, observation, and failure recovery on real execution data.

| Model | Routine Execution | | Observation-dependent | | Failure Recovery | | Critical Decisions | | Overall | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Tool | Call | Tool | Call | Tool | Call | Obs. Call | Repair Call | Tool | Call |
| *API model* | | | | | | | | | | |
| GPT-6 Astra | 86.8 | 79.6 | 78.2 | 69.4 | 73.8 | 63.5 | 64.4 | 56.7 | 79.5 | 70.8 |
| *Embodied models* | | | | | | | | | | |
| Embodied-R1.5 | 55.4 | 45.4 | 44.3 | 31.7 | 40.4 | 28.3 | 22.2 | 20.0 | 46.7 | 35.1 |
| RoboBrain2.5-8B | 51.9 | 40.8 | 38.1 | 26.0 | 35.4 | 22.7 | 16.7 | 13.3 | 41.8 | 29.8 |
| RynnBrain11-9B | 52.2 | 42.0 | 41.0 | 28.1 | 35.7 | 23.5 | 20.0 | 15.0 | 42.9 | 31.1 |
| Molmo2-ER-4B | 49.6 | 36.4 | 37.9 | 25.4 | 34.5 | 21.3 | 15.6 | 11.7 | 40.7 | 27.7 |
| *Open-source models* | | | | | | | | | | |
| Qwen3.5-9B | 55.8 | 45.0 | 42.8 | 30.3 | 38.2 | 26.4 | 20.0 | 13.3 | 45.6 | 33.9 |
| Qwen3.5-9B + harness | 62.2 | 52.9 | 59.3 | 48.9 | 55.4 | 44.3 | 46.7 | 41.7 | 58.9 | 48.6 |
| $\Delta$ vs. base (pp) | +6.4 | +7.9 | +16.5 | +18.6 | +17.2 | +17.9 | +26.7 | +28.4 | +13.3 | +14.7 |
| Qwen3.8-27B | 63.8 | 54.1 | 52.6 | 41.8 | 49.2 | 37.0 | 28.9 | 23.3 | 55.2 | 44.3 |
| Qwen3.8-27B + harness | 73.8 | 64.8 | 68.8 | 60.0 | 65.0 | 54.4 | 56.7 | 51.7 | 69.1 | 59.6 |
| $\Delta$ vs. base (pp) | +10.0 | +10.7 | +16.2 | +18.2 | +15.8 | +17.4 | +27.8 | +28.4 | +13.9 | +15.3 |

without physical operations, also seen in direct execution and potentially reinforced by harness evidence and verification. Exploration limits and action commitment prevent budget exhaustion.

**Explicit recovery supports anomaly handling.** Without recovery, failure-recovery SR drops from 39.2% to 27.7% on ProcTHOR and 65.4% to 21.5% on BEHAVIOR, while routine execution and observation-dependent performance remain largely stable or improve slightly. This selective decline supports verification and recovery for identifying unmet action effects and organizing corrective actions, with benefits concentrated in anomaly handling.

**Memory supports observation-dependent decisions.** Removing Evidence Memory lowers observation-dependent SR from 37.8% to 33.3% on ProcTHOR and from 34.4% to 17.8% on BEHAVIOR. Composite SR also falls from 30.0% to 23.0% on BEHAVIOR, with a smaller change on ProcTHOR. This supports retaining evidence for subsequent observation-dependent decisions.

### 5.4 Real-Observation Evaluation

We test transfer of simulation findings through offline next-tool-call evaluation on 200 distinct AgiBotWorld2026 (AgiBot, 2026) demonstration episodes: 80 normal-execution, 60 observation-dependent, and 60 involving anomalous or incomplete actions and repair. At each node, direct execution and SCOPER-harness predict the next tool call from the instruction, current semantic action segment's head-camera images, and preceding reference actions; predictions are not propagated as executed actions. We report tool accuracy and full-call accuracy (correct tool and all required arguments), separately evaluating observation-critical and repair-decision nodes. This fixed-history evaluation tests real-visual decisions, but not active observation acquisition, consequences of agents' own errors, or sustained multi-step recovery. It complements closed-loop simulation testing task completion through continued interaction. Construction and scoring appear in Appendix D. Consistent with simulation (Table 3), GPT-6 Astra leads overall full-call accuracy at 70.8%, while embodied-specialized models show no consistent advantage. SCOPER-harness raises full-call accuracy from 33.9% to 48.6% for Qwen3.5-9B and 44.3% to 59.6% for Qwen3.8-27B. Observation-critical call accuracy improves by 26.7/27.8 percentage points, respectively, and repair-call accuracy by 28.4 points for both, benefiting real visual decisions, especially critical nodes.

## 6 Conclusion

We introduced the SCOPER suite for studying embodied System 2 through closed-loop, observation-grounded planning and recovery. SCOPER-bench combines high-level goals, execution-time observations, and controlled failures in 870 tasks across two simulators. SCOPER-harness connects accumulated scene evidence and verified action outcomes to an evolving task agenda, improving performance across the evaluated models and on real-observation decision tests. These findings highlight the importance of using observations to determine required work and revise decisions throughout execution. Future work will extend the current skill interface toward richer, more continuous action spaces and broader action coverage, and adapt the benchmark and harness to more complex scenes with greater diversity, dynamics, and interaction demands.

### AI USE STATEMENT

We used Codex and the models GPT-5.6-Sol, GPT-6-Astra, and Gemini-3.8-flash for AI-assisted research discussions and code implementation. AI assistance also supported manuscript drafting and revision, literature search and reference checking, and figure-script editing. The benchmark additionally uses AI models as evaluated agents, as described in the experimental setup. The authors directed these uses, reviewed and revised the assisted content, and take responsibility for the final paper and its scientific claims.

### REPRODUCIBILITY STATEMENT

Section 3 describes benchmark construction and evaluation, and Section 4 presents the harness. The appendices document task templates and dataset statistics (Appendix G), execution infrastructure and controlled failures (Appendices E and F), evaluation protocols and situated QA (Appendices A and I), and harness state definitions and pseudocode (Appendix J). Real-observation evaluation details are provided in Appendix D. Additional supporting files will be provided as supplementary material.

## REFERENCES

AgiBot. AgiBot World 2026. Dataset, 2026. URL https://huggingface.co/datasets/Alleneworld/AgiBotWorld2026. Accessed September 24, 2026.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. *arXiv preprint arXiv:2204.01691*, 2022. doi: 10.48550/arXiv.2204.01691. URL https://arxiv.org/abs/2204.01691.

Michael Ahn, Debidatta Dwibedi, Chelsea Finn, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Karol Hausman, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Sean Kirmani, Isabel Leal, Edward Lee, Sergey Levine, Yao Lu, Sharath Maddineni, Kanishka Rao, Dorsa Sadigh, Pannag Sanketi, Pierre Sermanet, Quan Vuong, Stefan Welker, Fei Xia, Ted Xiao, Peng Xu, Steve Xu, and Zhuo Xu. AutoRT: Embodied Foundation Models for Large Scale Orchestration of Robotic Agents. *arXiv preprint arXiv:2401.12963*, 2024. doi: 10.48550/arXiv.2401.12963. URL https://arxiv.org/abs/2401.12963.

Yi Chen, Yuying Ge, Yixiao Ge, Mingyu Ding, Bohao Li, Rui Wang, Ruifeng Xu, Ying Shan, and Xihui Liu. EgoPlan-Bench: Benchmarking Multimodal Large Language Models for Human-Level Planning. *arXiv preprint arXiv:2312.06722*, 2023. doi: 10.48550/arXiv.2312.06722. URL https://arxiv.org/abs/2312.06722.

Sijie Cheng, Zhicheng Guo, Jingwen Wu, Kechen Fang, Peng Li, Huaping Liu, and Yang Liu. EgoThink: Evaluating First-Person Perspective Thinking Capability of Vision-Language Models. *arXiv preprint arXiv:2311.15596*, 2023. doi: 10.48550/arXiv.2311.15596. URL https://arxiv.org/abs/2311.15596.

Sijie Cheng, Kechen Fang, Yangyang Yu, Sicheng Zhou, Bohao Li, Ye Tian, Tingguang Li, Lei Han, and Yang Liu. VidEgoThink: Assessing Egocentric Video Understanding Capabilities for Embodied AI. *arXiv preprint arXiv:2410.11623*, 2024. doi: 10.48550/arXiv.2410.11623. URL https://arxiv.org/abs/2410.11623.

Zhili Cheng, Yuge Tu, Ran Li, Shiqi Dai, Jinyi Hu, Shengding Hu, Jiahao Li, Yang Shi, Tianyu Yu, Weize Chen, Lei Shi, and Maosong Sun. EmbodiedEval: Evaluate Multimodal LLMs as Embodied Agents. *arXiv preprint arXiv:2501.11858*, 2025. doi: 10.48550/arXiv.2501.11858. URL https://arxiv.org/abs/2501.11858.

Ronghao Dang, Yuqian Yuan, Wenqi Zhang, Yifei Xin, Boqiang Zhang, Long Li, Liuyi Wang, Qinyang Zeng, Xin Li, and Lidong Bing. ECBench: Can Multi-modal Foundation Models Understand the Egocentric World? A Holistic Embodied Cognition Benchmark. *arXiv preprint arXiv:2501.05031*, 2025. doi: 10.48550/arXiv.2501.05031. URL https://arxiv.org/abs/2501.05031.

Matt Deitke, Winson Han, Alvaro Herrasti, Aniruddha Kembhavi, Eric Kolve, Roozbeh Mottaghi, Jordi Salvador, Dustin Schwenk, Eli VanderBilt, Matthew Wallingford, Luca Weihs, Mark Yatskar, and Ali Farhadi. RoboTHOR: An Open Simulation-to-Real Embodied AI Platform. *arXiv preprint arXiv:2004.06799*, 2020. doi: 10.48550/arXiv.2004.06799. URL https://arxiv.org/abs/2004.06799.

Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Jordi Salvador, Kiana Ehsani, Winson Han, Eric Kolve, Ali Farhadi, Aniruddha Kembhavi, and Roozbeh Mottaghi. ProcTHOR: Large-Scale Embodied AI Using Procedural Generation. *arXiv preprint arXiv:2206.06994*, 2022. doi: 10.48550/arXiv.2206.06994. URL https://arxiv.org/abs/2206.06994.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. PaLM-E: An Embodied Multimodal Language Model. *arXiv preprint arXiv:2303.03378*, 2023. doi: 10.48550/arXiv.2303.03378. URL https://arxiv.org/abs/2303.03378.

Haoquan Fang, Jiafei Duan, Donovan Clay, Sam Wang, Shuo Liu, Weikai Huang, Xiang Fan, Wei-Chuan Tsai, Shirui Chen, Yi Ru Wang, Shanli Xing, Jaemin Cho, Jae Sung Park, Ainaz Eftekhar, Peter Sushko, Karen Farley, Angad Wadhwa, Cole Harrison, Winson Han, Ying-Chun Lee, Eli VanderBilt, Rose Hendrix, Suveen Ellawela, Lucas Ngoo, Joyce Chai, Zhongzheng Ren, Ali Farhadi, Dieter Fox, and Ranjay Krishna. MolmoAct2: Action Reasoning Models for Real-world Deployment. *arXiv preprint arXiv:2605.02881*, 2026. doi: 10.48550/arXiv.2605.02881. URL https://arxiv.org/abs/2605.02881.

Google DeepMind. Gemini 3.8 Flash Model Card, 2026. URL https://deepmind.google/models/model-cards/gemini-3-8-flash/. Accessed September 24, 2026.

Kristen Grauman, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, Hao Jiang, Miao Liu, Xingyu Liu, Miguel Martin, Tushar Nagarajan, Ilija Radosavovic, Santhosh Kumar Ramakrishnan, Fiona Ryan, Jayant Sharma, Michael Wray, Mengmeng Xu, Eric Zhongcong Xu, Chen Zhao, Siddhant Bansal, Dhruv Batra, Vincent Cartillier, Sean Crane, Tien Do, Morrie Doulaty, Akshay Erapalli, Christoph Feichtenhofer, Adriano Fragomeni, Qichen Fu, Abrham Gebreselasie, Cristina Gonzalez, James Hillis, Xuhua Huang, Yifei Huang, Wenqi Jia, Weslie Khoo, Jachym Kolar, Satwik Kottur, Anurag Kumar, Federico Landini, Chao Li, Yanghao Li, Zhenqiang Li, Karttikeya Mangalam, Raghava Modhugu, Jonathan Munro, Tullie Murrell, Takumi Nishiyasu, Will Price, Paola Ruiz Puentes, Merey Ramazanova, Leda Sari, Kiran Somasundaram, Audrey Southerland, Yusuke Sugano, Ruijie Tao, Minh Vo, Yuchen Wang, Xindi Wu, Takuma Yagi, Ziwei Zhao, Yunyi Zhu, Pablo Arbelaez, David Crandall, Dima Damen, Giovanni Maria Farinella, Christian Fuegen, Bernard Ghanem, Vamsi Krishna Ithapu, C. V. Jawahar, Hanbyul Joo, Kris Kitani, Haizhou Li, Richard Newcombe, Aude Oliva, Hyun Soo Park, James M. Rehg, Yoichi Sato, Jianbo Shi, Mike Zheng Shou, Antonio Torralba, Lorenzo Torresani, Mingfei Yan, and Jitendra Malik. Ego4D: Around the World in 3,000 Hours of Egocentric Video. *arXiv preprint arXiv:2110.07058*, 2021. doi: 10.48550/arXiv.2110.07058. URL https://arxiv.org/abs/2110.07058.

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. *arXiv preprint arXiv:2311.18259*, 2023. doi: 10.48550/arXiv.2311.18259. URL https://arxiv.org/abs/2311.18259.

Qiao Gu, Alihusein Kuwajerwala, Sacha Morin, Krishna Murthy Jatavallabhula, Bipasha Sen, Aditya Agarwal, Corban Rivera, William Paul, Kirsty Ellis, Rama Chellappa, Chuang Gan, Celso Miguel de Melo, Joshua B. Tenenbaum, Antonio Torralba, Florian Shkurti, and Liam Paull. ConceptGraphs: Open-Vocabulary 3D Scene Graphs for Perception and Planning. *arXiv preprint arXiv:2309.16650*, 2023. doi: 10.48550/arXiv.2309.16650. URL https://arxiv.org/abs/2309.16650.

Yanjiang Guo, Yen-Jen Wang, Lihan Zha, and Jianyu Chen. DoReMi: Grounding Language Model by Detecting and Recovering from Plan-Execution Misalignment. *arXiv preprint arXiv:2307.00329*, 2023. doi: 10.48550/arXiv.2307.00329. URL https://arxiv.org/abs/2307.00329.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/TSSC.1968.300136. URL https://doi.org/10.1109/TSSC.1968.300136.

Chenguang Huang, Oier Mees, Andy Zeng, and Wolfram Burgard. Visual Language Maps for Robot Navigation. *arXiv preprint arXiv:2210.05714*, 2022a. doi: 10.48550/arXiv.2210.05714. URL https://arxiv.org/abs/2210.05714.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. *arXiv preprint arXiv:2201.07207*, 2022b. doi: 10.48550/arXiv.2201.07207. URL https://arxiv.org/abs/2201.07207.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner Monologue: Embodied Reasoning through Planning with Language Models. *arXiv preprint arXiv:2207.05608*, 2022c. doi: 10.48550/arXiv.2207.05608. URL https://arxiv.org/abs/2207.05608.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models. *arXiv preprint arXiv:2307.05973*, 2023. doi: 10.48550/arXiv.2307.05973. URL https://arxiv.org/abs/2307.05973.

Krishna Murthy Jatavallabhula, Alihusein Kuwajerwala, Qiao Gu, Mohd Omama, Tao Chen, Alaa Maalouf, Shuang Li, Ganesh Iyer, Soroush Saryazdi, Nikhil Keetha, Ayush Tewari, Joshua B. Tenenbaum, Celso Miguel de Melo, Madhava Krishna, Liam Paull, Florian Shkurti, and Antonio Torralba. ConceptFusion: Open-set Multimodal 3D Mapping. *arXiv preprint arXiv:2302.07241*, 2023. doi: 10.48550/arXiv.2302.07241. URL https://arxiv.org/abs/2302.07241.

Baoxiong Jia, Ting Lei, Song-Chun Zhu, and Siyuan Huang. EgoTaskQA: Understanding Human Tasks in Egocentric Videos. *arXiv preprint arXiv:2210.03929*, 2022. doi: 10.48550/arXiv.2210.03929. URL https://arxiv.org/abs/2210.03929.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, Aniruddha Kembhavi, Abhinav Gupta, and

Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv preprint arXiv:1712.05474*, 2017. doi: 10.48550/arXiv.1712.05474. URL https://arxiv.org/abs/1712.05474.

Chengshu Li, Fei Xia, Roberto Martín-Martín, Michael Lingelbach, Sanjana Srivastava, Bokui Shen, Kent Vainio, Cem Gokmen, Gokul Dharan, Tanish Jain, Andrey Kurenkov, C. Karen Liu, Hyowon Gweon, Jiajun Wu, Li Fei-Fei, and Silvio Savarese. iGibson 2.0: Object-Centric Simulation for Robot Learning of Everyday Household Tasks. *arXiv preprint arXiv:2108.03272*, 2021. doi: 10.48550/arXiv.2108.03272. URL https://arxiv.org/abs/2108.03272.

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Wensi Ai, Benjamin Martinez, Hang Yin, Michael Lingelbach, Minjune Hwang, Ayano Hiranaka, Sujay Garlanka, Arman Aydin, Sharon Lee, Jiankai Sun, Mona Anvari, Manasi Sharma, Dhruva Bansal, Samuel Hunter, Kyu-Young Kim, Alan Lou, Caleb R Matthews, Ivan Villa-Renteria, Jerry Huayang Tang, Claire Tang, Fei Xia, Yunzhu Li, Silvio Savarese, Hyowon Gweon, C. Karen Liu, Jiajun Wu, and Li Fei-Fei. BEHAVIOR-1K: A Human-Centered, Embodied AI Benchmark with 1,000 Everyday Activities and Realistic Simulation. *arXiv preprint arXiv:2403.09227*, 2024. doi: 10.48550/arXiv.2403.09227. URL https://arxiv.org/abs/2403.09227.

Kehan Li, Bohan Hou, Minghao Zhu, Tianyi Zhang, Zesen Cheng, Zhikai Wang, Sicong Leng, Xin Li, Xiao Lin, Biying Yao, Minghua Zeng, Jiangpin Liu, Ronghao Dang, Jiayan Guo, Siteng Huang, Haoyu Zhao, Heng Ping, Yaxi Zhao, Tong Zhao, Kexiang Wang, Tong Lu, Shengke Xue, Jiahao Tang, Yulei Wang, Zejing Wang, Jianwei Gao, Shijian Lu, Chengju Liu, Jianfei Yang, Mingxiu Chen, and Deli Zhao. RynnBrain 1.1: Towards More Capable and Generalizable Embodied Foundation Model. *arXiv preprint arXiv:2607.17977*, 2026. doi: 10.48550/arXiv.2607.17977. URL https://arxiv.org/abs/2607.17977.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as Policies: Language Model Programs for Embodied Control. *arXiv preprint arXiv:2209.07753*, 2022. doi: 10.48550/arXiv.2209.07753. URL https://arxiv.org/abs/2209.07753.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking Knowledge Transfer for Lifelong Robot Learning. *arXiv preprint arXiv:2306.03310*, 2023a. doi: 10.48550/arXiv.2306.03310. URL https://arxiv.org/abs/2306.03310.

Zeyi Liu, Arpit Bahety, and Shuran Song. REFLECT: Summarizing Robot Experiences for Failure Explanation and Correction. *arXiv preprint arXiv:2306.15724*, 2023b. doi: 10.48550/arXiv.2306.15724. URL https://arxiv.org/abs/2306.15724.

Arjun Majumdar, Anurag Ajay, Xiaohan Zhang, Pranav Putta, Sriram Yenamandra, Mikael Henaff, Sneha Silwal, Paul Mcvay, Oleksandr Maksymets, Sergio Arnaud, Karmesh Yadav, Qiyang Li, Ben Newman, Mohit Sharma, Vincent Berges, Shiqi Zhang, Pulkit Agrawal, Yonatan Bisk, Dhruv Batra, Mrinal Kalakrishnan, Franziska Meier, Chris Paxton, Sasha Sax, and Aravind Rajeswaran. OpenEQA: Embodied question answering in the era of foundation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Majumdar_OpenEQA_Embodied_Question_Answering_in_the_Era_of_Foundation_Models_CVPR_2024_paper.html.

Karttikeya Mangalam, Raiymbek Akshulakov, and Jitendra Malik. EgoSchema: A Diagnostic Benchmark for Very Long-form Video Language Understanding. *arXiv preprint arXiv:2308.09126*, 2023. doi: 10.48550/arXiv.2308.09126. URL https://arxiv.org/abs/2308.09126.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. CALVIN: A Benchmark for Language-Conditioned Policy Learning for Long-Horizon Robot Manipulation Tasks. *arXiv preprint arXiv:2112.03227*, 2021. doi: 10.48550/arXiv.2112.03227. URL https://arxiv.org/abs/2112.03227.

Yao Mu, Qinglong Zhang, Mengkang Hu, Wenhai Wang, Mingyu Ding, Jun Jin, Bin Wang, Jifeng Dai, Yu Qiao, and Ping Luo. EmbodiedGPT: Vision-Language Pre-Training via Embodied Chain of Thought. *arXiv preprint arXiv:2305.15021*, 2023. doi: 10.48550/arXiv.2305.15021. URL https://arxiv.org/abs/2305.15021.

OpenAI. GPT-6 Astra System Card. Technical report, OpenAI, 2026. URL https://deploymentsafety.openai.com/gpt-6-astra. Accessed September 24, 2026.

Aishwarya Padmakumar, Jesse Thomason, Ayush Shrivastava, Patrick Lange, Anjali Narayan-Chen, Spandana Gella, Robinson Piramuthu, Gokhan Tur, and Dilek Hakkani-Tur. TEACh: Task-driven Embodied Agents that Chat. *arXiv preprint arXiv:2110.00534*, 2021. doi: 10.48550/arXiv.2110.00534. URL https://arxiv.org/abs/2110.00534.

Jiaqi Peng, Xiqian Yu, Delin Feng, Yuqiang Yang, Wenzhe Cai, Jing Xiong, Ganlin Yang, Jinliang Zheng, Jiafei Cao, Xueyuan Wei, Jiangmiao Pang, Yuan Shen, and Tai Wang. Cortex: A Bidirectionally Aligned Embodied Agent Framework for Long-horizon Manipulation. *arXiv preprint arXiv:2607.05377*, 2026. doi: 10.48550/arXiv.2607.05377. URL https://arxiv.org/abs/2607.05377.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. VirtualHome: Simulating Household Activities via Programs. *arXiv preprint arXiv:1806.07011*, 2018. doi: 10.48550/arXiv.1806.07011. URL https://arxiv.org/abs/1806.07011.

Xavier Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Tsung-Yen Yang, Ruslan Partsey, Ruta Desai, Alexander William Clegg, Michal Hlavac, So Yeon Min, Vladimír Vondruš, Theophile Gervet, Vincent-Pierre Berges, John M. Turner, Oleksandr Maksymets, Zsolt Kira, Mrinal Kalakrishnan, Jitendra Malik, Devendra Singh Chaplot, Unnat Jain, Dhruv Batra, Akshara Rai, and Roozbeh Mottaghi. Habitat 3.0: A Co-Habitat for Humans, Avatars and Robots. *arXiv preprint arXiv:2310.13724*, 2023. doi: 10.48550/arXiv.2310.13724. URL https://arxiv.org/abs/2310.13724.

Lu Qiu, Yi Chen, Yuying Ge, Yixiao Ge, Ying Shan, and Xihui Liu. EgoPlan-Bench2: A Benchmark for Multimodal Large Language Model Planning in Real-World Scenarios. *arXiv preprint arXiv:2412.04447*, 2024. doi: 10.48550/arXiv.2412.04447. URL https://arxiv.org/abs/2412.04447.

Qwen Team. Qwen3.5-9B model card. Hugging Face model repository, 2026a. URL https://huggingface.co/Qwen/Qwen3.5-9B. Accessed September 24, 2026.

Qwen Team. Qwen3.8-27B model card. Hugging Face model repository, 2026b. URL https://huggingface.co/Qwen/Qwen3.8-27B. Accessed September 24, 2026.

Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Robot Task Planning. *arXiv preprint arXiv:2307.06135*, 2023. doi: 10.48550/arXiv.2307.06135. URL https://arxiv.org/abs/2307.06135.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. *arXiv preprint arXiv:1904.01201*, 2019. doi: 10.48550/arXiv.1904.01201. URL https://arxiv.org/abs/1904.01201.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. *arXiv preprint arXiv:2303.11366*, 2023. doi: 10.48550/arXiv.2303.11366. URL https://arxiv.org/abs/2303.11366.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. *arXiv preprint arXiv:1912.01734*, 2019. doi: 10.48550/arXiv.1912.01734. URL https://arxiv.org/abs/1912.01734.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. *arXiv preprint arXiv:2010.03768*, 2020. doi: 10.48550/arXiv.2010.03768. URL https://arxiv.org/abs/2010.03768.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating Situated Robot Task Plans using Large Language Models. *arXiv preprint arXiv:2209.11302*, 2022. doi: 10.48550/arXiv.2209.11302. URL https://arxiv.org/abs/2209.11302.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models. *arXiv preprint arXiv:2212.04088*, 2022. doi: 10.48550/arXiv.2212.04088. URL https://arxiv.org/abs/2212.04088.

Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Martín-Martín, Fei Xia, Kent Vainio, Zheng Lian, Cem Gokmen, Shyamal Buch, C. Karen Liu, Silvio Savarese, Hyowon Gweon, Jiajun Wu, and Li Fei-Fei. BEHAVIOR: Benchmark for Everyday Household Activities in Virtual, Interactive, and Ecological Environments. *arXiv preprint arXiv:2108.03332*, 2021. doi: 10.48550/arXiv.2108.03332. URL https://arxiv.org/abs/2108.03332.

Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training Home Assistants to Rearrange their Habitat. *arXiv preprint arXiv:2106.14405*, 2021. doi: 10.48550/arXiv.2106.14405. URL https://arxiv.org/abs/2106.14405.

Huajie Tan, Enshen Zhou, Zhiyu Li, Yijie Xu, Yuheng Ji, Xiansheng Chen, Cheng Chi, Pengwei Wang, Huizhu Jia, Yulong Ao, Mingyu Cao, Sixiang Chen, Zhe Li, Mengzhen Liu, Zixiao Wang, Shanyu Rong, Yaoxu Lyu, Zhongxia Zhao, Peterson Co, Yibo Li, Yi Han, Shaoxuan Xie, Guocai Yao, Songjing Wang, Leiduo Zhang, Xi Yang, Yance Jiao, Donghai Shi, Kunchang Xie, Shaokai Nie, Chunlei Men, Yonghua Lin, Zhongyuan Wang, Tiejun Huang, and Shanghang Zhang. RoboBrain 2.5: Depth in Sight, Time in Mind. *arXiv preprint arXiv:2601.14352*, 2026. doi: 10.48550/arXiv.2601.14352. URL https://arxiv.org/abs/2601.14352.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv preprint arXiv:2305.16291*, 2023a. doi: 10.48550/arXiv.2305.16291. URL https://arxiv.org/abs/2305.16291.

Qi Wang, Tianyi Wang, Chengyang Li, Shikun Ban, Yurun Chen, Yizhong Ge, Jason Qin, Chengtai Li, and Wentao Zhu. Towards the Harness of Embodied Agents. *arXiv preprint arXiv:2608.11246*, 2026. doi: 10.48550/arXiv.2608.11246. URL https://arxiv.org/abs/2608.11246.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents. *arXiv preprint arXiv:2302.01560*, 2023b. doi: 10.48550/arXiv.2302.01560. URL https://arxiv.org/abs/2302.01560.

Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. TidyBot: Personalized Robot Assistance with Large Language Models. *arXiv preprint arXiv:2305.05658*, 2023a. doi: 10.48550/arXiv.2305.05658. URL https://arxiv.org/abs/2305.05658.

Zhenyu Wu, Ziwei Wang, Xiuwei Xu, Jiwen Lu, and Haibin Yan. Embodied Task Planning with Large Language Models. *arXiv preprint arXiv:2307.01848*, 2023b. doi: 10.48550/arXiv.2307.01848. URL https://arxiv.org/abs/2307.01848.

Haoming Xu, Zhenlin He, Hengyi Wang, Jiafeng Xu, and Hao Dong. Mimir: A Neuro-Symbolic Memory System with Dynamic Grounding for Embodied Agents in Interactive Environments. *arXiv preprint arXiv:2608.04933*, 2026. doi: 10.48550/arXiv.2608.04933. URL https://arxiv.org/abs/2608.04933.

Jingkang Yang, Shuai Liu, Hongming Guo, Yuhao Dong, Xiamengwei Zhang, Sicheng Zhang, Pengyun Wang, Zitang Zhou, Binzhu Xie, Ziyue Wang, Bei Ouyang, Zhengyu Lin, Marco Cominelli, Zhongang Cai, Yuanhan Zhang, Peiyuan Zhang, Fangzhou Hong, Joerg Widmer, Francesco Gringoli, Lei Yang, Bo Li, and Ziwei Liu. EgoLife: Towards Egocentric Life Assistant. *arXiv preprint arXiv:2503.03803*, 2025a. doi: 10.48550/arXiv.2503.03803. URL https://arxiv.org/abs/2503.03803.

Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, Heng Ji, Huan Zhang, and Tong Zhang. EmbodiedBench: Comprehensive Benchmarking Multi-modal Large Language Models for Vision-Driven Embodied Agents. *arXiv preprint arXiv:2502.09560*, 2025b. doi: 10.48550/arXiv.2502.09560. URL https://arxiv.org/abs/2502.09560.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. *arXiv preprint arXiv:2210.03629*, 2022. doi: 10.48550/arXiv.2210.03629. URL https://arxiv.org/abs/2210.03629.

Sriram Yenamandra, Arun Ramachandran, Karmesh Yadav, Austin Wang, Mukul Khanna, Theophile Gervet, Tsung-Yen Yang, Vidhi Jain, Alexander William Clegg, John Turner, Zsolt Kira, Manolis Savva, Angel Chang, Devendra Singh Chaplot, Dhruv Batra, Roozbeh Mottaghi, Yonatan Bisk, and Chris Paxton. HomeRobot: Open-Vocabulary Mobile Manipulation. *arXiv preprint arXiv:2306.11565*, 2023. doi: 10.48550/arXiv.2306.11565. URL https://arxiv.org/abs/2306.11565.

Yifu Yuan, Yaoting Huang, Xianze Yao, Yutong Li, Shuoheng Zhang, Linqi Han, Pengyi Li, Jiangeng Sun, Wenting Jia, Zhao Zhang, Yuhao Liu, Ruihao Liao, Yucheng Hu, Qiyu Wu, Yuxiao Li, Zibin Dong, Fei Ni, Yan Zheng, Shuyang Gu, Yi Ma, Hongyao Tang, Han Hu, and Jianye Hao. Embodied-R1.5: Evolving Physical Intelligence via Embodied Foundation Models. *arXiv preprint arXiv:2606.11324*, 2026. doi: 10.48550/arXiv.2606.11324. URL https://arxiv.org/abs/2606.11324.

Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, and Pete Florence. Socratic Models: Composing Zero-Shot Multimodal Reasoning with Language. *arXiv preprint arXiv:2204.00598*, 2022. doi: 10.48550/arXiv.2204.00598. URL https://arxiv.org/abs/2204.00598.

# APPENDIX GUIDE

The links below lead to each appendix section.


- Appendix A: Evaluation Protocol
- Appendix B: High-Level Action Counts
- Appendix C: Blind Evaluation
- Appendix D: Real-Robot Next-Tool-Call Evaluation
- Appendix E: Oracle System 1 and Teacher Policy
- Appendix F: Controlled Failure Injection and Scene Disturbances
- Appendix G: Task Templates and Construction
- Appendix H: Binding Procedure
- Appendix I: Situated QA
- Appendix J: Harness Details
- Appendix K: Execution Trace Visualization
- Appendix L: Harness Model-Call Overhead


# A EVALUATION PROTOCOL

Evaluation compares closed-loop execution through a shared skill interface and task-level scorer. This section specifies episode interaction, available information, and aggregation; detailed QA definitions appear in Appendix I.

## A.1 INTERACTION, OBSERVATIONS, AND INFORMATION ACCESS

Each episode loads a task, initializes the scene and event schedulers, and produces initial RGB images. The model emits one skill request from its current decision context; Binding resolves public references, System 1 executes admitted requests, and the interface supplies images and request records for the next decision. A separate evaluator checks goals and events.

All evaluated methods receive public instructions, tool definitions, RGB images, and permitted request records. Simulator inventories and identifiers, role mappings, ground-truth states and poses, distances, goal progress, injection labels, teacher trajectories, and reference decompositions remain internal. Public vocabulary defines valid references rather than scene contents. Harness memory, agenda, and visual assessments derive from public inputs. Execution-admission signals are used by the deterministic controller only for gating and budget accounting; executor success flags and diagnostic records are excluded from model inputs.

Navigation, scanning, and target search each produce up to eight process-observation frames, retaining the current view; ordinary manipulation supplies before-and-after RGB images. These observations enter a visual history maintained across actions. Direct evaluation models receive the latest available frames, up to 32, together with public action-request history at each decision. The harness uniformly samples up to eight frames from navigation, scanning, and search for observation maintenance; its policy uses up to the latest four frames, public request history, and routed memory. Since memory retains evidence from earlier observations, the policy uses fewer recent frames than direct evaluation. When fewer frames are available, all available frames within the respective limit are used. Instructions, public vocabulary, and tool definitions remain accessible throughout execution.

The harness uses Qwen3.5-9B and Qwen3.8-27B; policy, verification, memory, and agenda calls within an experiment use the same fixed-parameter base model. Additional model calls contribute computational cost rather than additional environmental observations; acquiring a different viewpoint or position requires a public tool request and consumes the corresponding interaction budget.

Direct-model evaluation uses an input image resolution of $512 \times 512$; GPT-6 uses the `max` reasoning-effort setting.

### A.2 Evaluation Prompts and Per-Decision Context

The fixed system prompt assigns the model the role of an embodied agent completing the task with the listed skills. It supplies the instruction, tool names and argument definitions, public reference rules, and general action prerequisites. Instructions specify objectives, scope, and conditional requirements. Shared guidance covers approaching objects before interaction, assessing effects from images, observing again when needed, and stopping after all task clauses are satisfied; the model chooses the task-specific action sequence.

Each user turn contains the current step, remaining tool budget, the previous requested skill and public arguments, chronologically ordered action-request history, and the current image input. The request-history limit follows the episode's high-level budget, capped at 48 entries, allowing preceding requests to be retained within the prescribed 24-, 36-, or 48-step episodes. Each entry states what was requested without an action-success label. Full user–assistant conversations are not replayed by default, and old images are not repeatedly embedded in the history text; previously generated reasoning is not retained as permanent environmental evidence.

The model outputs one tool name and its arguments, optionally accompanied by a short rationale. The interface parses the request before Binding and execution. The next-turn summary repeats the requested call and states that it has finished; the model assesses its outcome from images.

The following layout distinguishes fixed interaction rules from turn-specific content. Images are attached chronologically as image inputs; dynamic content comes from the public task and interaction records. New task messages enter subsequent task context, whereas QA messages use a dedicated answer format.

> **System:** Complete the task using the available tools. Task: [instruction and public task updates]. Tools: [names and argument definitions]. Follow the public reference rules and interaction prerequisites. Use images to assess outcomes; a requested action does not establish success. Return one tool call with arguments.
>
> **Decision turn:** Step: [current step]. Remaining budget: [remaining calls]. Previous requests: [public action history]. Last request: [tool and arguments; call finished]. Visual input: [ordered frames]. Choose the next call.
>
> **QA turn:** Question: [identifier, text, and choices]. Use the permitted current context and return the question identifier and selected answer value.

When a situated question is posted, a separate answer call supplies its public identifier, text, choices, and the visual and execution context permitted at that time; the model responds with the question identifier and selected answer value. Newly introduced task requirements are supplied as public natural-language messages and distinguished from questions requesting only an answer. After stopping, the prompt specifies an answer-only phase without further physical actions; goal checking and answer correctness remain evaluator-side operations.

### A.3 Budgets and Termination

Atomic, two-subtask, and three-subtask episodes use high-level call budgets of 24, 36, and 48, respectively, shared across compared methods. Observation, manipulation, answering, and stopping are counted under a common protocol; appended QA answers also consume this budget. Internal verification, memory maintenance, and action reselection calls are recorded separately, enabling separate comparisons of environment-interaction efficiency and model-call cost.

Under the standard stop-and-QA finishing protocol, `stop` ends physical execution and freezes its task score. Pending questions are handled in an answer-only finishing phase using the remaining budget, without resuming physical actions. Physical completion and subsequent answers can therefore be scored separately.

Task completion is determined from environment state at scoring time and applicable process conditions, rather than a model's completion claim or earlier goal attainment. Let $\mathcal{G}_i$ denote an episode's goal predicates. We set $s_i = 1$ exactly when all predicates hold and the task's configured stopping requirement is satisfied. A stop call freezes the goal score; if the budget expires without stopping, the last available state is evaluated and any unmet stopping requirement remains unsatisfied. Budget

exhaustion is additionally recorded as a runner failure, so runner status and physical task success are reported separately.

For composite tasks, handoff records local child completion, whereas overall scoring uses the parent task's required goal set; handoff records do not replace the overall goal check. Event records separately establish whether an intervention was triggered and applied. An untriggered intervention provides no evidence of successful recovery. Goal scores, intervention application, and QA scores remain distinct.

### A.4 Metrics and Aggregation

We compute task metrics over a common set $\mathcal{D}$ of scored episodes, including both successful tasks and agent failures during valid evaluation. Infrastructure errors and unfinished runs are recorded separately rather than counted as task successes or failures. Task-group and template results use the corresponding subsets; proportions below are reported as percentages.

**Success Rate (SR).** SR measures the fraction of episodes satisfying their final success conditions:

$$\mathrm{SR} = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} s_i, \qquad s_i \in \{0, 1\}. \tag{1}$$

Here, $s_i$ is the evaluator's `task_success` indicator, requiring the scored state to satisfy the task goals, including applicable process conditions, and stopping conditions. It is neither the model's completion claim nor merely a successful runner termination.

**Task Progress (Prog.).** Progress measures partial completion. Let $\mathcal{G}_i$ be the goal predicates of episode $i$ and $g_{ij} \in \{0, 1\}$ indicate whether predicate $j$ holds at scoring time. We first compute the satisfied fraction within each episode and then average over episodes:

$$\mathrm{Prog.} = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \frac{\sum_{j \in \mathcal{G}_i} g_{ij}}{|\mathcal{G}_i|}. \tag{2}$$

Thus, tasks with more predicates do not automatically receive greater weight.

Goal items can be environmental predicates or evaluator-recorded process conditions. Container inspection uses two goal items: a valid interior inspection and the condition-dependent final openness. Logical branches of that condition are not separate scoring items.

**Necessary Action Coverage (Cov.).** Coverage measures how many required actions from the teacher policy's reference execution are covered. For each task, a validated teacher trajectory is compiled into a deduplicated reference action set with dependency constraints $\mathcal{R}_i$, which serves as the ground-truth reference for this metric. Matching uses the normalized tool, the resolved execution target, and the manipulated object for placement or release. Same-object causal dependencies and composite-stage constraints are retained, while unrelated actions may be reordered. Only execution-confirmed matches count; matching request names or repeated attempts alone do not increase coverage. References remain private to the evaluator.

Let $\mathcal{C}_i \subseteq \mathcal{R}_i$ contain the matched reference actions. Coverage is normalized within each task before averaging:

$$\mathrm{Cov.} = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \frac{|\mathcal{C}_i|}{|\mathcal{R}_i|}. \tag{3}$$

This measures execution coverage relative to the teacher reference, rather than shortest-path efficiency or final goal correctness, and is reported separately from SR and progress.

QA aggregation includes only questions with valid evaluator answer keys. Questions whose keys cannot be resolved are recorded separately and excluded from the corresponding configured and posted counts, as well as correct-answer counts. Missing, incorrect, or unparseable model responses are not answer-key resolution failures and remain incorrect responses to valid posted questions.

Table 4: Average high-level calls over all scored episodes, including agent failures. Lower values do not necessarily indicate better efficiency.

| Model | ProcTHOR | BEHAVIOR |
|---|---|---|
| GPT-6 Astra | 23.07 | 22.45 |
| Embodied-R1.5 | 12.88 | 14.46 |
| RoboBrain2.5-8B | 10.08 | 12.54 |
| RynnBrain11-9B | 21.88 | 21.21 |
| Molmo2-ER-4B | 18.08 | 18.31 |
| Qwen3.5-9B | 20.06 | 19.45 |
| Qwen3.5-9B + Harness | 22.91 | 22.29 |
| Qwen3.8-27B | 20.68 | 21.68 |
| Qwen3.8-27B + Harness | 22.94 | 22.41 |

**QA Trigger Rate and Triggered QA Accuracy (T / A).** These metrics distinguish reaching question checkpoints from answering them correctly. Let $q_i$, $p_i$, and $c_i$ denote the numbers of valid configured, valid posted, and correctly answered situated questions in episode $i$. The main tables report

$$T = \frac{\sum_{i \in \mathcal{D}} p_i}{\sum_{i \in \mathcal{D}} q_i}, \qquad A = \frac{\sum_{i \in \mathcal{D}} c_i}{\sum_{i \in \mathcal{D}} p_i}. \tag{4}$$

Questions eligible for activation but still awaiting delivery in the queue do not count as posted.

Triggered but unanswered questions count as incorrect in $A$; untriggered questions are excluded from its denominator but reduce $T$. A metric with a zero denominator is reported as unavailable. Different execution paths can expose different question sets, so accuracy is interpreted alongside trigger rate. These metrics concern situated QA.

**Average Steps.** Let $L_i$ be the number of high-level calls counted by the interaction protocol in episode $i$. We report

$$\text{AvgSteps} = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} L_i. \tag{5}$$

Both successful and failed episodes are included. Observation, navigation, manipulation, answering, and stopping follow the shared counting protocol; low-level simulator actions and internal harness model calls are excluded. Atomic, two-subtask, and three-subtask budgets are 24, 36, and 48 calls, respectively. Fewer steps can result from early failure, so this metric is not designated lower-is-better.

**Aggregation.** SR, progress, coverage, and average steps are weighted by episode counts; QA metrics are weighted by their respective question denominators. We do not directly average template or category percentages or assign equal weight to per-episode QA accuracies. Overall results are computed from underlying counts or equivalent denominator-weighted aggregates.

## B High-Level Action Counts

Table 4 reports average high-level calls; these should be interpreted jointly with task completion. On ProcTHOR, RoboBrain2.5-8B uses 10.08 steps with 5.8% success, whereas Qwen3.8-27B uses 20.68 steps and achieves 21.1% success. Shorter trajectories therefore do not necessarily imply more effective execution.

## C Blind Evaluation

To verify that SCOPER-bench requires visual observations rather than textual shortcuts, we replace all visual inputs with black images. Tables 5 and 6 report the results for BEHAVIOR and ProcTHOR, respectively. All models exhibit near-zero task completion, relying largely on prior-based guesses. These results demonstrate that the benchmark requires visual perception and cannot be solved through simple instruction decomposition.

Table 5: BEHAVIOR blind testing by task group (%).

| Model | Routine Execution | | | Observation-dependent | | | Failure Recovery | | | Composite | | | Overall | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SR | Prog. | Cov. | SR | Prog. | Cov. | SR | Prog. | Cov. | SR | Prog. | Cov. | SR | Prog. | Cov. |
| *API model* | | | | | | | | | | | | | | | |
| GPT-6 Astra | 0.0 | 22.2 | 37.5 | 1.1 | 18.5 | 34.8 | 0.8 | 28.5 | 48.1 | 2.0 | 29.5 | 30.1 | 1.0 | 25.1 | 38.5 |
| Gemini-3.8-flash | 0.0 | 14.5 | 24.7 | 1.1 | 10.9 | 20.4 | 0.8 | 17.8 | 30.0 | 1.0 | 15.5 | 15.7 | 0.7 | 15.0 | 23.3 |
| *Embodied models* | | | | | | | | | | | | | | | |
| Embodied-R1.5 | 2.0 | 24.8 | 16.0 | 7.8 | 20.6 | 25.6 | 8.5 | 18.9 | 23.5 | 3.0 | 21.8 | 13.0 | 5.5 | 21.3 | 19.6 |
| RoboBrain2.5-8B | 1.0 | 20.6 | 11.0 | 3.3 | 9.7 | 18.3 | 4.6 | 14.2 | 16.8 | 2.0 | 20.6 | 6.2 | 2.9 | 16.3 | 13.2 |
| RynnBrain11-9B | 1.0 | 22.5 | 11.4 | 4.4 | 8.6 | 10.4 | 3.1 | 11.4 | 13.8 | 2.0 | 18.0 | 3.6 | 2.6 | 15.0 | 10.1 |
| Molmo2-ER-4B | 0.0 | 21.4 | 6.4 | 1.1 | 3.7 | 5.9 | 6.2 | 17.3 | 9.8 | 2.0 | 14.7 | 2.9 | 2.6 | 14.7 | 6.5 |
| *Open-source models* | | | | | | | | | | | | | | | |
| Qwen3.5-9B | 2.0 | 24.9 | 14.8 | 10.0 | 14.9 | 16.8 | 5.4 | 16.0 | 19.6 | 2.0 | 20.3 | 7.6 | 4.8 | 18.9 | 15.0 |
| Qwen3.5-9B + harness | 2.0 | 23.4 | 28.5 | 4.4 | 16.5 | 27.2 | 1.5 | 23.3 | 36.5 | 2.0 | 25.8 | 21.1 | 2.4 | 22.5 | 28.9 |
| Qwen3.8-27B | 5.0 | 24.9 | 21.5 | 5.6 | 17.4 | 25.9 | 7.7 | 23.2 | 27.4 | 3.0 | 21.7 | 14.2 | 5.5 | 22.0 | 22.5 |
| Qwen3.8-27B + harness | 1.0 | 22.7 | 34.2 | 2.2 | 18.4 | 33.2 | 2.3 | 27.5 | 43.5 | 2.0 | 28.1 | 27.1 | 1.9 | 24.5 | 35.2 |

Table 6: ProcTHOR blind testing by task group (%).

| Model | Routine Execution | | | Observation-dependent | | | Failure Recovery | | | Composite | | | Overall | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SR | Prog. | Cov. | SR | Prog. | Cov. | SR | Prog. | Cov. | SR | Prog. | Cov. | SR | Prog. | Cov. |
| *API model* | | | | | | | | | | | | | | | |
| GPT-6 Astra | 4.6 | 38.6 | 43.4 | 3.3 | 35.4 | 46.6 | 3.8 | 35.1 | 51.4 | 2.0 | 32.2 | 36.2 | 3.6 | 35.5 | 44.7 |
| Gemini-3.8-flash | 2.3 | 24.4 | 27.4 | 3.3 | 24.0 | 31.1 | 2.3 | 21.6 | 32.0 | 1.0 | 20.4 | 23.6 | 2.2 | 22.6 | 28.6 |
| *Embodied models* | | | | | | | | | | | | | | | |
| Embodied-R1.5 | 1.5 | 23.0 | 6.2 | 5.6 | 12.6 | 15.0 | 3.1 | 14.5 | 14.8 | 1.0 | 22.2 | 4.9 | 2.7 | 18.3 | 10.1 |
| RoboBrain2.5-8B | 0.0 | 16.4 | 4.1 | 3.3 | 9.9 | 8.9 | 3.1 | 12.8 | 8.5 | 2.0 | 15.6 | 6.4 | 2.0 | 13.9 | 6.8 |
| RynnBrain11-9B | 1.5 | 22.6 | 5.8 | 2.2 | 8.8 | 6.7 | 0.8 | 18.8 | 8.6 | 0.0 | 14.9 | 5.4 | 1.1 | 17.0 | 6.7 |
| Molmo2-ER-4B | 0.0 | 18.4 | 3.2 | 1.1 | 3.7 | 2.4 | 0.0 | 10.6 | 2.9 | 1.0 | 10.7 | 1.1 | 0.4 | 11.5 | 2.5 |
| *Open-source models* | | | | | | | | | | | | | | | |
| Qwen3.5-9B | 0.8 | 22.7 | 11.6 | 6.7 | 16.1 | 15.4 | 2.3 | 19.3 | 19.5 | 2.0 | 24.1 | 11.9 | 2.7 | 20.7 | 14.7 |
| Qwen3.5-9B + harness | 3.8 | 31.8 | 31.2 | 4.4 | 28.5 | 35.2 | 2.3 | 28.0 | 37.3 | 2.0 | 28.3 | 26.6 | 3.1 | 29.3 | 32.7 |
| Qwen3.8-27B | 2.3 | 29.9 | 29.5 | 4.4 | 25.0 | 33.2 | 3.8 | 30.6 | 33.3 | 3.0 | 33.3 | 25.7 | 3.3 | 29.9 | 30.5 |
| Qwen3.8-27B + harness | 4.6 | 37.0 | 40.8 | 3.3 | 32.7 | 43.3 | 3.8 | 34.6 | 48.7 | 2.0 | 32.6 | 33.6 | 3.6 | 34.5 | 42.0 |

# D Real-Robot Next-Tool-Call Evaluation

## D.1 Data and Decision-Node Construction

The evaluation uses real-robot demonstrations from AgiBotWorld2026, excluding simulation data. The 200 distinct source episodes form three mutually exclusive groups: 80 normal-execution, 60 observation-dependent, and 60 involving anomalous or incomplete actions and their repair. Each sample covers a contiguous task scope with a defined objective and termination condition within one source video; videos are not spliced together. An episode can yield multiple decision nodes, so episode counts differ from scored-decision counts.

Videos are segmented into semantic actions such as navigation, picking, placing, and opening or closing containers. Local camera, torso, and gripper adjustments are generally absorbed into their semantic segment, whereas necessary base approaches are represented as navigation. For reference calls $(a_1, \ldots, a_K)$, the node after segment $i$ receives the reference prefix through $a_i$ and targets $a_{i+1}$; the final segment is followed by `finish()`. Scoring begins after the first segment and does not predict the initial action. Every node uses reference history rather than predictions from preceding nodes.

### D.2 Observations and Challenge Categories

Visual inputs use the head-camera stream `observation.images.top_head`, sampled at 0.5 FPS on a fixed temporal grid anchored at the start of the selected task scope. Each node receives only grid frames within the current segment and no later than its endpoint, without accumulating raw images from earlier segments. Textual action history retains the complete reference prefix. Future frames, manually selected keyframes, bounding-box annotations, success or failure labels, category labels, and reference answers are withheld.

English instructions are reconstructed from source annotations and videos, specifying execution order where necessary. Observation-dependent instructions describe inspection requirements without revealing the object or state to discover; repair instructions do not identify a failed action. Each observation-dependent episode includes at least one node whose next call requires current visual evidence. Nodes requiring future images to resolve ambiguity are excluded from scoring.

Recovery samples include naturally occurring anomalies with verifiable subsequent recovery and incomplete-action nodes constructed by splitting an original action before completion. For example, a pickup segment can be divided into an unfinished pickup and its continuation, followed by placement. The latter preserves the original images but does not imply an actual grasp failure or controller retry. The two sources are labeled separately, and decisions that resume the task after repair are evaluated.

### D.3 Harness State Along Reference Execution

The harness maintains memory and the agenda along the reference execution prefix. The initial agenda is derived from the instruction, and state is updated using images and reference calls provided up to the current node. Raw image input still covers only the current segment; earlier observations can persist through memory. Predicted calls are not executed, do not enter reference history, and do not authorize committing expected effects. Subsequent updates follow the reference video and actions without future frames or answer labels.

### D.4 Output Interface and Metrics

Each node freely generates one structured call without candidate options or a complete future plan. Tools include `navigate_to`, `pick`, `place`, `open`/`close`, `turn_on`/`turn_off`, `soak_under`, and `finish`, with data-dependent extensions fixed before evaluation. Object arguments are matched at category level using a frozen alias mapping; instance identifiers, colors, coordinates, and hand selection are not evaluated. Placement specifies the destination without repeating the held object's category. The interface contains no `inspect` tool.

Tool Accuracy measures the fraction of nodes with the correct tool name. The primary metric, Call Accuracy, requires the tool and all necessary arguments to match one complete reference. Argument errors invalidate the call but not a correct tool name; unparseable outputs or unknown tools fail both metrics. The argument-free `finish()` receives identical judgments. Equivalent references must be matched as complete calls rather than by combining their components.

Category and overall results are aggregated over decision nodes; overall accuracy divides total correct predictions by all scorable nodes. Observation-critical Call Accuracy and Repair-decision Call Accuracy use their respective subsets without adding them again to the overall denominator. Post-repair continuation nodes contribute to the repair category. Because predictions do not control the robot or subsequent observations, this evaluation reports no task success, goal progress, action coverage, or average execution steps.

## E Oracle System 1 and Teacher Policy

Oracle System 1 is skill-execution infrastructure shared by evaluated models and the teacher. It implements individual skill requests through rules; the teacher policy reads the task specification and selects skill sequences for reference execution and validation. We describe skill execution and teacher organization separately below.

### E.1 Unified Skill Abstraction

System 1 converts high-level requests into controlled scene changes and provides visual observations of their execution and outcomes. Table 7 summarizes skills by shared semantics: navigation and observation establish information and interaction poses, while manipulation updates object poses, relations, or properties. Object-state processing can be realized by a dedicated skill or a composition of existing skills.

Table 7: Unified System 1 skills. Rows describe semantic capabilities rather than identical API names across backends.

| Skill | Preconditions and input | Controlled execution and observations |
|---|---|---|
| Navigation | Resolved object or room target; reachable interaction pose. | Follow an A* path, update the agent pose, and provide observations along the route. |
| Scan | Available viewpoints around the current pose. | Rotate the view and acquire multiple observations of the surroundings. |
| Target search | Public target reference. | Direct observation toward relevant views to obtain visual evidence about the target. |
| Pick | Pickupable, accessible, interaction-ready object. | Establish the held relation and make the object pose follow the executor. |
| Place / release | Held object; an admissible destination for placement. | Place at a support or containment pose, or release at the current position; remove the held relation and update spatial relations. |
| Open / close | Openable, accessible container. | Update openness and the corresponding joint or visual configuration. |
| Toggle on / off | Toggleable, interaction-ready device. | Change activation state and synchronize its appearance and applicable semantic effects. |
| Cleaning | Compatible object and required tool or water source. | Update cleanliness and its visual representation. |
| Slice / cut | Sliceable object with required tool and access conditions. | Update the sliced state and associated geometry or object identities. |
| Heat / cool | Compatible object and appropriately configured heat or cold source. | Update heated, cooked, or cooled states. |

### E.2 Tool Arguments and Model-Facing Outputs

Table 8 summarizes model-facing tool arguments. Unless noted otherwise, listed arguments are required strings; `object_ref` uses a public concrete category or supported alias rather than a private role or object identifier. Skills share semantics, with interface differences stated only where necessary: ProcTHOR includes room navigation in `navigate_to`, whereas BEHAVIOR additionally exposes room navigation and several state-processing tools. Models receive only the tool definitions available in the current backend.

Table 8: Public tool parameters and observation semantics.

| Tool | Arguments | Meaning and returned observation |
|---|---|---|
| `navigate_to` | `object_ref` | Object/receptacle, or a public room label in ProcTHOR; route and endpoint images. |
| `navigate_to_room` | `room` | Public room label in BEHAVIOR; process images. |
| `scan` | None. | Multiple in-place views obtained by rotating the camera. |
| `look_for` | `object_ref` | Target-directed camera views; does not replace approaching the object. |
| `pick_up` | `object_ref` | Object to pick up; before-and-after images. |

| Tool | Arguments | Meaning and returned observation |
|---|---|---|
| put_down | object_ref | Destination receptacle or support, not the held object; before-and-after images. |
| drop | None. | Release at the current position in BEHAVIOR; before-and-after images. |
| open, close, toggle_on, toggle_off | object_ref | Container or device to operate; before-and-after images. |
| slice / cut | object_ref | Food to cut, not the cutting tool; before-and-after images. |
| wipe | object_ref | Object to clean with a held tool; before-and-after images. |
| soak_under | object_ref | Running water source for cleaning the held item; before-and-after images. |
| fill | object_ref, source | Container and fluid source; before-and-after images. |
| pour | object_ref | Receiver for contents of the held container; before-and-after images. |
| heat, cool | object_ref | Heat/cold device receiving the held item; before-and-after images. |
| answer | question_id, value | Posted question identifier and selected answer text; no new scene evidence. |
| stop | None. | End physical execution; no goal-success oracle is returned. |

Wiping, washing under a source, filling, pouring, and dedicated heating and cooling tools are exposed by BEHAVIOR; corresponding ProcTHOR state effects arise through the container, device, and placement operations described above. State transitions occur inside System 1, while the model receives prescribed execution images and a request-completion summary, not state values, held-object inventories, ground-truth spatial relations, or effect labels. Answering and stopping are protocol operations and do not acquire additional scene information.

### E.3 Navigation and Observation Implementation

Navigation constructs its search space from scene collision and reachability information and uses A* to reach an interaction pose near the target. ProcTHOR searches a grid of reachable positions and executes movement and rotation along the path. BEHAVIOR searches an occupancy grid accounting for the agent footprint and validates connectivity and path feasibility; its eight-connected search disallows diagonal corner cutting through blocked cells.

Navigation, scanning, and target search provide images during execution through backend observation callbacks. Scanning covers surrounding views, while target search organizes target-directed observation. The evaluation interface selects and delivers observations to the model or harness, connecting scene discovery to action selection throughout execution.

### E.4 Manipulation and State Synchronization

Manipulation controls poses, attachment relations, and object properties through rules, bypassing low-level grasp and contact dynamics. ProcTHOR's controlled pickup and placement maintain visual attachment and containment relations alongside simulator state operations. BEHAVIOR's symbolic executor updates held states, spatial relations, and properties and synchronizes these changes with rendered objects and joints. Both retain object-affordance, access, and interaction preconditions.

State processing follows the semantics of the corresponding operation. For example, ProcTHOR implements cleaning through sink and faucet interactions and cooling or heating through container and device operations. BEHAVIOR uses corresponding state operations and execution wrappers for

cleaning, cutting, and temperature changes. A shared capability therefore need not correspond to an identically named standalone tool on both backends. The resulting scene and images provide evidence for subsequent outcome checking.

Manipulation skills use controlled state transitions that bypass low-level motion and contact simulation while retaining immediate pre- and post-action RGB observations for outcome checking. Navigation and exploration additionally provide multi-frame process observations, allowing the model to discover objects and conditions during execution.

The teacher policy uses task specifications and environment truth to organize skill calls into reference trajectories. Routine strategies satisfy goal predicates, observation-dependent strategies handle scene issues to be discovered, recovery strategies address execution deviations, and composite strategies coordinate subtasks. Reference trajectories support feasibility, goal, and event validation and provide execution context for QA attachment. Evaluated models execute through the same System 1 skill interface.

# F Controlled Failure Injection and Scene Disturbances

Controlled interventions turn routine skill execution into closed-loop tasks requiring monitoring and recovery. We distinguish execution failures that alter the current skill's outcome from scene disturbances that change the environment after an execution milestone. Both are specified by activation conditions and state transformations in the task configuration, allowing the same task goal to be paired with different recovery challenges.

## F.1 Activation and Scheduling

We represent an event as $e = (\tau, r, h, \phi, n)$, where $\tau$ specifies an eligible skill or milestone, $r$ identifies the target role, $h$ defines activation conditions, $\phi$ specifies the state intervention, and $n$ bounds the number of firings. Conditions can depend on a particular action result, elapsed action steps, containment or non-held states, and the owning subtask being active. Configurations also support probabilities and cooldown intervals to control intervention frequency.

Execution failures intercept a matching skill call or modify its outcome, whereas scene disturbances check conditions and apply changes after the relevant execution milestone. The scheduler maintains event histories and firing counts and scopes events to their owning subtask. Eligibility and whether the intended state change was actually applied are recorded separately for task validation.

## F.2 Failure Mechanisms

Table 9 summarizes six concrete failure mechanisms. They alter holding, placement, openness, or navigation outcomes, requiring subsequent actions to be revised from actual observations. For example, a dropped placement releases the object outside its destination; recovery requires localization and pickup before another placement attempt.

Table 9: Execution failures: activation, state consequences, and recovery requirements.

| Mechanism | Trigger and realized consequence | Recovery requirement |
|---|---|---|
| Grasp slip | A pickup releases the object at a recoverable nearby location. | Relocalize, approach, and pick again. |
| Dropped placement | Placement releases the object beside the destination, violating the required relation. | Reacquire the object and place it again. |
| Partial opening | An open request leaves insufficient openness. | Inspect and complete opening. |
| Partial closing | A close request leaves the container partly open. | Inspect and complete closing. |
| Wrong-target navigation | Navigation executes toward an alternative object. | Detect the mismatch and approach the intended target. |
| Incomplete navigation | The final agent pose falls short of the required interaction position. | Reassess the current pose and navigate onward. |

Object release and relocation use controlled pose and held-relation updates and refresh subsequent observations. Dropped placement selects a location outside the destination; BEHAVIOR addition-

ally checks collision, reachability, and rendered visibility and restores the prior state if the relocation cannot be committed. Partial opening and closing modify openness and its visual configuration. Wrong-target navigation substitutes the executed target while retaining the original request for outcome checking.

Incomplete navigation simulates failure to arrive by modifying the final pose. ProcTHOR repositions the agent to an intermediate location between the start and endpoint after nominal navigation; BEHAVIOR selects a pre-terminal pose from the planned route. This mechanism creates an outcome requiring further navigation without adding a physical obstacle to the scene.

### F.3 State-Changing Disturbances

Scene disturbances can reclose or reopen containers, toggle devices, re-dirty objects, and relocate or restore objects. Activation connects these changes to meaningful task stages: for example, moving a staged object once it is no longer held, or re-dirtying an object after cleaning and intervening actions. Such events require the model to revisit previously valid facts and restore disrupted prerequisites or goals.

Both backends implement disturbances through scene-state changes and synchronize the visual representation of object poses, joints, or properties. Task validation checks final goals together with required interventions, ensuring that reference trajectories encounter the intended challenge. The evaluator retains private event records, while models assess consequences from protocol-provided observations and public interaction information.

## G Task Templates and Construction

Templates translate capability requirements into instantiable object, state, and event constraints. We organize tasks into Routine Execution, Observation-dependent, Failure Recovery, and Composite. Below we describe shared template fields, instantiation, and representative instances. Tables below summarize each benchmark template through its roles, initialization requirements, and goal predicates.

### G.1 Dataset Statistics and Scene Sampling

The construction target is ten task instances per base template in each simulator, together with 50 two-child and 50 three-child composite tasks per simulator. The 35 base templates thus yield a target of 450 tasks per backend. Limited matches across diverse BEHAVIOR scenes lead to the exclusion of 30 instances, leaving 420 BEHAVIOR and 450 ProcTHOR tasks, or 870 tasks in total. ProcTHOR contains 130 routine-execution, 90 observation-dependent, 130 failure-recovery, and 100 composite episodes. BEHAVIOR contains 100, 90, 130, and 100 episodes in the corresponding groups, with the 30 excluded instances belonging to routine execution. ProcTHOR scenes are randomly sampled from its official pool of 12,000 scenes. For BEHAVIOR, filtering out excessively large scenes leaves a pool of 35 distinct scenes for task matching; task objects are randomly added under the template's compatibility constraints.

### G.2 Template Specification and Instantiation

Each template specifies object roles, categories or category groups, affordance requirements, and relations between roles. Initial conditions describe open/closed, cleanliness, containment, or location states, and goal predicates define the desired outcome. Instruction patterns replace roles with public object descriptions; event configurations define challenge triggers and consequences; composite tasks additionally retain subtask identities and scopes.

Scene matching first filters candidates by object type and affordance, then checks same-room, cross-room, distinctness, and spatial constraints. ProcTHOR profiles provide object and scene information. The BEHAVIOR matcher can reuse the shared JSON templates and distinguishes direct matches, matches requiring spawned assets, and matches requiring runtime validation. Bound tasks satisfy initial conditions through permitted relocation, state configuration, or asset materialization.

Initialization applies these configurations to concrete objects and synchronizes rendered state. Validation checks initial predicates, geometry and support relations, navigation and sequential interaction reachability, and whether all goals already hold. Teacher rollouts then check final goals and required events. Scenes unable to support a template retain explicit rejection reasons.

Instructions specify task objectives, scope, and handling rules without exposing construction-time instance bindings or step-by-step procedures. In observation-conditioned tasks, the model identifies the concrete objects requiring attention and their current states from scene observations. Public category terms express selection criteria rather than enumerate qualifying scene instances. The model consequently determines action targets, necessary prerequisites, and subsequent operations.

For tasks with universal inspection requirements, construction ensures that all actionable issues within the instruction scope are represented in template bindings and scoring goals; other relevant objects are initialized in compliant states. Instructions such as switching everything off or closing all containers therefore introduce no additional initial issues outside the scoring scope.

For example, clean-dishware storage requires dishware, storage, and distinguishable dirty-dishware distractors. The matcher selects instances satisfying affordance and spatial constraints; initialization makes target dishware clean, makes distractor dishware dirty, and configures placement under the template constraints. The public instruction requests storing clean dishware and excluding dirty dishware from storage, without identifying which concrete instances belong to either group or prescribing a pickup-and-placement sequence. Final goals jointly require target storage, exclusion of distractors from storage, and container closure where applicable.

### G.3 Representative Templates

Routine execution includes simple placement and preparation before storage. Simple placement transfers a pickupable object to a destination; preparation requires cleaning, cutting, heating, or cooling before the placement goal. Multi-stage preparation combines these requirements into an object workflow.

The dirty-dish workflow template includes dishware, sink, faucet, and storage roles and can include a staging surface. The dish is initialized dirty and outside storage, and the goal requires it to be clean and stored. The task connects identifying issues in the scene to subsequent cleaning and storage, making observations determine which objects and operations require attention.

Safety inspection initializes a running faucet and an open container in one room and requires both to be closed or switched off. Other observation templates include cross-room inspection, mixed issues, and category-dependent storage. Latent issue records specify the actual initial conditions and required handling, while the model identifies actionable conditions from visual observations.

The dropped-placement template binds a pickupable object and destination, with a goal of establishing the correct placement relation. A one-shot placement event releases the object beside the target, requiring recovery through approach, pickup, and return to the destination. Partial opening or closing, grasp slips, and incorrect navigation alter other execution effects, requiring different recovery procedures toward the same final goal.

Composite tasks contain two or three subtasks and retain each child's roles, goals, events, and identity. Runtime scheduling scopes events to their owning subtask and advances the active child at handoff. Handoffs are managed by the evaluation process rather than requested by the model. QA can cover different children and handoff progress; overall and per-subtask success are recorded separately.

### G.4 Template Structure and Predicate Catalog

Table 10 summarizes template structure. Table 11 describes the 35 benchmark task templates individually through object roles, initialization or event requirements, and goal predicates. Roles satisfy the affordance and access constraints of their operations; the tables highlight spatial conditions that distinguish task semantics.

For a unified notation, $P(x, r)$ denotes the placement relation required by a template, instantiated as containment or surface support; $P(X, r)$ requires the relation for every target in the set.

Table 10: Semantic structure of a task template.

| Component | Specification |
|---|---|
| Roles | Object categories, affordances, optionality, and inter-role relations. |
| Initialization | Object properties, initial containment and placement, and task-relevant initial visibility. |
| Goals | Object-state and spatial-relation predicates, including conditional or set-valued goals. |
| Instruction | Public descriptions grounded in bound roles and observation-conditioned handling rules. |
| Events and QA | Activation conditions, state interventions, questions, and scoring rules. |
| Composition | Child identities, scopes, and handoff relations. |

$\bigwedge_{d \in D} \neg P(d, r)$ requires each distractor to remain outside the destination. $C^*(r)$ requires closure only when the destination is openable. Optional roles contribute goals when instantiated. Event requirements define the challenge alongside initial states but activate during execution.

The benchmark uses 35 task templates. Routine Execution, Observation-dependent, Failure Recovery, and Composite are capability-level groups, and an individual template can involve several capabilities. Initially running faucets or dirty dishes are issues to discover through observation, whereas execution-failure templates alter outcomes during skill calls. Composite tasks are formed by composing the selected templates.

Table 11: Benchmark task templates: roles, initialization, and goal predicates.

| Template | Roles and initialization / events | Goal predicates |
|---|---|---|
| Simple placement | Object $x$, receptacle $r$ in the same room.<br>$x$ visible; $\neg P(x, r)$. | $P(x, r)$. |
| Device activation | Toggleable device $d$.<br>$\neg\mathrm{On}(d)$. | $\mathrm{On}(d)$. |
| Clean and place | Dirtyable $x$, sink $w$, faucet $f$, destination $r$.<br>$\mathrm{Dirty}(x)$; compatible sink–faucet pair. | $\mathrm{Clean}(x) \wedge P(x, r) \wedge C^*(r)$. |
| Heat and place | Object $x$, microwave $h$, destination $r$.<br>$\neg\mathrm{On}(h)$. | $\mathrm{Hot}(x) \wedge P(x, r) \wedge \mathrm{Closed}(h) \wedge \neg\mathrm{On}(h)$. |
| Cool and place | Object $x$, refrigerator $h$, destination $r \neq h$.<br>Compatible cooling and placement roles. | $\mathrm{Cool}(x) \wedge P(x, r) \wedge \mathrm{Closed}(h)$. |
| Slice and place | Sliceable $x$, knife $k$, destination $r$.<br>Compatible slicing and placement roles. | $\mathrm{Sliced}(x) \wedge P(x, r)$. |
| Slice, cool, and place | $x$, knife $k$, work surface $w$, refrigerator $h$, destination $r$.<br>Roles co-located; distinct processing and destination roles. | $\mathrm{Sliced}(x) \wedge \mathrm{Cool}(x) \wedge P(x, r)$. |
| Wash, heat, and store | $x$, sink $w$, faucet $f$, microwave $h$, storage $r$.<br>$\mathrm{Dirty}(x)$. | $\mathrm{Clean}(x) \wedge \mathrm{Hot}(x) \wedge P(x, r) \wedge C^*(r)$. |
| Store dishware | Dishware set $X$, storage $r$, optional distractors $D$.<br>Mandatory target outside storage; optional relocation support. | $P(X, r) \wedge \bigwedge_{d \in D} \neg P(d, r) \wedge C^*(r)$. |

| Template (continued) | Roles and initialization / events | Goal predicates |
|---|---|---|
| Store cleaning supplies | Cleaning supplies $X$, storage $r$, optional distractors $D$.<br>Mandatory target outside storage; optional relocation support. | $P(X, r) \wedge \bigwedge_{d \in D} \neg P(d, r) \wedge C^*(r)$. |
| Select clean dishware | Clean dishware $X$, dirty distractor $d$, storage $r$.<br>Mandatory targets clean; $\mathrm{Dirty}(d)$; optional extra target. | $P(X, r) \wedge \neg P(d, r) \wedge C^*(r)$. |
| Store food | Food set $X$, cold storage $r$, optional distractors $D$.<br>Mandatory target outside storage. | $P(X, r) \wedge \bigwedge_{d \in D} \neg P(d, r) \wedge \mathrm{Closed}(r)$. |
| Cross-room food storage | Food $x$, refrigerator $r$, source-room anchor.<br>$\neg P(x, r)$; source and destination in different rooms. | $P(x, r) \wedge C^*(r)$. |
| Explore and place | Object $x$, destination $r$.<br>$x$ outside the initial view. | $P(x, r) \wedge C^*(r)$. |
| Cross-room inspection | Faucet $f$, container $c$ in separate rooms.<br>$\mathrm{On}(f) \wedge \mathrm{Open}(c)$; initially unseen issues. | $\neg \mathrm{On}(f) \wedge \mathrm{Closed}(c)$. |
| Dirty-dish workflow | Dish $x$, sink $w$, faucet $f$, storage $r$, optional staging surface.<br>$\mathrm{Dirty}(x) \wedge \neg P(x, r)$. | $\mathrm{Clean}(x) \wedge P(x, r)$. |
| Mixed-issue inspection | Dish $x$, sink, faucet $f$, storage $r$, container $c$.<br>$\mathrm{Dirty}(x) \wedge \mathrm{On}(f) \wedge \mathrm{Open}(c)$. | $\mathrm{Clean}(x) \wedge P(x, r) \wedge \neg \mathrm{On}(f) \wedge \mathrm{Closed}(c)$. |
| Safety inspection | Faucet $f$, container $c$ in one room.<br>$\mathrm{On}(f) \wedge \mathrm{Open}(c)$. | $\neg \mathrm{On}(f) \wedge \mathrm{Closed}(c)$. |
| Storage-policy inspection | Misplaced object $x$, category-compatible storage $r$.<br>$\neg P(x, r)$. | $P(x, r)$. |
| Policy-based tidying | Sampled issue objects; optional faucet $f$.<br>Scene-specific violations materialized by the generator. | Conjunction of instantiated issue goals; $\neg \mathrm{On}(f)$ when present. |
| Place and handle dirty dish | Primary $x, r$; dirty dish $y$, sink, faucet, storage $s$.<br>$\mathrm{Dirty}(y)$. | $P(x, r) \wedge \mathrm{Clean}(y) \wedge P(y, s) \wedge C^*(r) \wedge C^*(s)$. |
| Handle faucet and dirty dish | Faucet $f$, sink, dish $x$.<br>$\mathrm{On}(f) \wedge \mathrm{Dirty}(x)$. | $\neg \mathrm{On}(f) \wedge \mathrm{Clean}(x)$. |
| Food storage with faucet issue | Food $x$, refrigerator $r$, faucet $f$.<br>$\mathrm{On}(f)$. | $P(x, r) \wedge \mathrm{Closed}(r) \wedge \neg \mathrm{On}(f)$. |
| Placement with open-container issue | Primary $x, r$; separate container $c$.<br>$\mathrm{Open}(c)$; issue container initially unseen. | $P(x, r) \wedge \mathrm{Closed}(c) \wedge C^*(r)$. |
| Dropped-placement recovery | Object $x$, destination $r$.<br>Inject drop beside $r$ on placement. | $P(x, r)$. |
| Grasp-slip recovery | Accessible pickupable $x$, destination $r$.<br>Inject slip on pickup. | $P(x, r)$. |
| Incomplete-navigation recovery | Object $x$, destination $r$.<br>Inject incomplete navigation toward $x$. | $P(x, r)$. |
| Wrong-navigation recovery | Object $x$, distractor $y$, destination $r$.<br>Redirect navigation from $x$ to $y$. | $P(x, r)$. |

| Template (continued) | Roles and initialization / events | Goal predicates |
|---|---|---|
| Partial-close recovery | Openable container $c$.<br>$\mathrm{Open}(c)$; inject partial closing. | $\mathrm{Closed}(c)$. |
| Partial-open recovery | Openable container $c$.<br>$\mathrm{Closed}(c)$; inject partial opening. | $\mathrm{Open}(c)$. |
| Destination reclosure | Object $x$, destination container $r$.<br>$\neg P(x,r) \wedge \mathrm{Closed}(r)$; reclose during execution. | $P(x,r) \wedge \mathrm{Closed}(r)$. |
| Object relocation | Object $x$, destination $r$, faucet interruption role.<br>Relocate the object during execution. | $P(x,r)$. |
| Cleanliness reversal | Dirtyable $x$, sink, faucet $f$, destination $r$.<br>$\mathrm{Dirty}(x) \wedge \neg\mathrm{On}(f)$; re-dirty during execution. | $\mathrm{Clean}(x) \wedge P(x,r)$. |
| Inspect and restore a storage container | Container $c$ and optional dishware $D$.<br>Vary initial contents and openness independently; require a valid observation of the interior. | $\mathrm{Inspected}(c) \wedge [(H_0(c) \wedge \mathrm{Closed}(c)) \vee (\neg H_0(c) \wedge \mathrm{Open}(c))]$. |
| Placement with added instruction | Object $x$, destination $r$, light $l$.<br>$\neg P(x,r) \wedge \mathrm{On}(l)$; a runtime instruction adds a requirement. | $P(x,r) \wedge C^*(r) \wedge \neg\mathrm{On}(l)$. |

Here, $H_0(c)$ records whether dishware is present in the container at initialization; moving contents during execution does not change the branch. $\mathrm{Inspected}(c)$ records acquisition of an interior observation sufficient to assess the contents, rather than merely requesting an observation tool. Construction validation requires visually discriminable evidence for both presence and absence; an occluded view does not establish inspection. The evaluator records whether an interior view delivered through the public observation interface satisfies the task's visual inspection condition; invoking a tool without obtaining such a view does not satisfy it. This template permits initial openness to satisfy the state requirement, but the inspection condition is initially unmet, so the complete task still requires execution. This prevents initial openness from determining the correct branch.

### G.5 Programmatic Instantiation and Composition

The five observation-conditioned inspection and workflow templates instantiate scene issues through programmatic recipes. Cross-room inspection configures one faucet or container issue in each of two rooms; safety inspection configures faucet and container issues in one room; mixed inspection configures faucet, container, and dirty-dish issues in one room. The dirty-dish workflow recipe configures one dish issue, and the storage-policy recipe configures one misplaced object. Each issue expands into concrete roles, initial states, and handling goals.

The general long-horizon constructor combines two or three subtasks while retaining their roles, events, and goal ownership. Compatible object and state requirements allow the children to share an episode while preserving their individual task specifications. The overall goal collects the required child outcomes, while events and QA are scheduled within child scopes.

### G.6 Task Instructions and Execution Mechanisms

Table 12 pairs representative public instructions with categories and execution mechanisms for 35 templates: 13 routine, nine observation-dependent, and 13 recovery templates. Conditional container inspection belongs to the observation-dependent group. Composite tasks combine atomic templates and are not counted again as atomic entries.

Categories follow the same template-family grouping as the result tables, rather than mutually exclusive capability requirements. For example, routine clean-dishware selection still requires visual

judgment; device activation and placement with an added instruction belong to the observation-dependent group. Failure Recovery includes four `recover_*` templates that address initially present anomalies, six skill-outcome failure mechanisms, and three mid-execution disturbances. The mechanism column describes construction settings and is not additional model input.

Object names illustrate possible scene instantiations, and instructions are semantic examples rather than verbatim evaluation prompts. Event triggers and parameters remain task-specific; an intervention is not implied for an entry without a listed disturbance. Situated QA can be attached separately without changing the underlying category.

Table 12: Representative task instructions, template categories, and key execution mechanisms. R: routine; O: observation-dependent; F: failure recovery. Category and mechanism annotations are not shown to the model.

| Category | Template | Representative instruction | Key execution mechanism |
|---|---|---|---|
| R | Simple placement | Put the mug on the table. | Transfer the object to its destination. |
| O | Device activation | Turn on the desk lamp. | Device initially off; activate it. |
| R | Clean and place | Wash the bowl and put it in the cabinet. Close the cabinet when finished. | Object initially dirty; establish cleanliness and storage. |
| R | Heat and place | Heat the potato in the microwave, then put it on the plate. Leave the microwave closed and switched off. | Heating must precede final placement; leave the microwave off and closed. |
| R | Cool and place | Cool the apple in the refrigerator, then move it to the table. Leave the refrigerator closed. | Cooling must precede transfer to a different destination. |
| R | Slice and place | Slice the apple and put it on the plate. | Slicing requires a compatible cutting tool. |
| R | Slice, cool, and place | Slice the apple, cool it in the refrigerator, and then put it on the plate. | Combine cutting, cooling, and final placement. |
| R | Wash, heat, and store | Wash the potato, heat it, and put it in the cabinet. Close the cabinet when finished. | Maintain cleaning and heating requirements through storage. |
| R | Store dishware | Put away the dishware in this room in the cabinet. Do not put other items in the cabinet. Close the cabinet when finished. | Select dishware from the scene; exclude distractors from storage. |
| R | Store cleaning supplies | Put away the cleaning supplies in this room in the cabinet. Do not put other items in the cabinet. Close the cabinet when finished. | Select cleaning supplies; exclude unrelated objects. |
| R | Select clean dishware | Inspect the dishware in this room and put only the clean items in the cabinet. Do not put dirty dishware in the cabinet. Close the cabinet when finished. | Distinguish clean targets from dirty distractors visually. |
| R | Store food | Put the food items in this room in the refrigerator. Do not put non-food items in the refrigerator. Close the refrigerator when finished. | Select food for cold storage; exclude non-food distractors. |
| R | Cross-room food storage | Move the apple from the dining room into the kitchen refrigerator. Close the refrigerator when finished. | Source object and destination are in different rooms. |
| O | Explore and place | Find the mug and put it in the cabinet. Close the cabinet when finished. | Target object starts outside the initial view. |

| Category | Template | Representative instruction | Key execution mechanism |
|---|---|---|---|
| O | Cross-room inspection | Inspect the kitchen and bathroom. Ensure all switches are off and all containers are closed. | Initially running faucet and open container occur in different rooms. |
| O | Dirty-dish workflow | Inspect the kitchen. Wash any dirty items you find and put them in their designated storage. | Discover initially dirty dishware and complete washing and storage. |
| O | Mixed-issue inspection | Inspect the kitchen. Ensure all switches are off and all containers are closed. Wash any dirty items you find and put them in their designated storage. | Discover coexisting faucet, container, and dirty-dish issues. |
| O | Safety inspection | Inspect this room and ensure all switches are off and all containers are closed. | Discover an initially running faucet and open container in one room. |
| O | Storage-policy inspection | Inspect this room for misplaced items and put them away according to these rules: food belongs in the refrigerator, and dishware belongs in the cabinet. | Identify misplaced objects and choose destinations by category. |
| R | Policy-based tidying | Inspect and tidy this room. Return misplaced items to their designated locations, address dirty dishware, and ensure that any faucets you encounter are off. | Scene-specific issues are instantiated from the tidying rules. |
| F | Place and handle dirty dish | Put the mug on the table. Also inspect the kitchen dishware and wash and store any items that need attention. Close any storage containers you use. | An initially dirty dish adds observation-dependent work to placement. |
| F | Handle faucet and dirty dish | Inspect the sink area, wash any dirty dishware, and ensure that the faucet is off. | Initially running faucet and dirty dish require inspection and handling. |
| F | Food storage with faucet issue | Put the apple in the refrigerator and close it. Also inspect the kitchen and ensure all switches are off and all containers are closed. | An initially running faucet must be addressed alongside food storage. |
| F | Placement with open-container issue | Put the mug on the table. Also inspect this room and close any open cabinets or drawers you find. | A separate container starts open and outside the initial view. |
| F | Dropped-placement recovery | Put the mug on the table. | Placement injection releases the object beside the destination; reacquire it. |
| F | Grasp-slip recovery | Put the mug on the table. | Pickup injection causes a slip and release; relocalize and pick again. |
| F | Incomplete-navigation recovery | Put the mug on the table. | Navigation ends short of the interaction pose; continue approaching. |
| F | Wrong-navigation recovery | Put the mug on the table. | Navigation is redirected to a distractor; approach the intended target. |
| F | Partial-close recovery | Close the cabinet. | Closing leaves the container partly open; inspect and finish closing. |
| F | Partial-open recovery | Open the cabinet. | Opening leaves insufficient openness; inspect and finish opening. |

| Category | Template | Representative instruction | Key execution mechanism |
|---|---|---|---|
| F | Destination reclosure | Put the mug in the cabinet and close the cabinet when finished. | A milestone-triggered disturbance recloses the destination container. |
| F | Object relocation | Put the mug on the table. | A milestone-triggered disturbance relocates the object; find it again. |
| F | Cleanliness reversal | Wash the bowl and put it on the table. | A disturbance makes the cleaned object dirty again; restore cleanliness. |
| O | Inspect and restore a storage container | Inspect the cabinet's contents. If it contains dishware, close it; otherwise, leave it open. | Vary contents and initial openness independently; require interior inspection and the corresponding final state. |
| O | Placement with added instruction | *Initial instruction:* Put the mug on the table. *Later message:* Also turn off the desk lamp before finishing. | A runtime message adds a new requirement to the initial placement task. |

Composite tasks preserve child goals and event scopes and can combine routine, observation-dependent, and recovery requirements.

# H Binding Procedure

Binding connects action requests expressed in public vocabulary to instance-level System 1 execution. It comprises construction-time role binding and runtime reference resolution: the former determines the task's concrete scene instances, while the latter converts the model's selected skill and object reference into an executable request. Both backends use shared resolution semantics and maintain simulator-instance mappings inside their adapters.

## H.1 Construction-Time Binding and Public References

During construction, template roles are bound to scene objects according to category, affordance, and spatial constraints. The internal mapping supports initialization, event scheduling, goal scoring, and runtime instance resolution. The model uses instructions and observations to select public object categories and operations, which the infrastructure resolves to concrete instances.

Tool instructions require object arguments to use a complete category name or an explicitly listed alias from the public vocabulary; room navigation uses public room labels from the instruction. Private roles, raw object identifiers, and invented color, position, or state modifiers are not valid references. Broad task concepts such as dishware or storage must be translated through observation into concrete categories supported by the interface.

## H.2 Runtime Resolution and Interaction Readiness

At runtime, the resolver normalizes the public reference and filters candidates according to reference-admission rules. Resolution first considers compatible recent interaction focus and task-designated instances, followed by public aliases, activated observation or memory candidates, and permitted scene or room references. Internal task mappings can therefore participate in instance selection among objects of the same category: a task- or subtask-designated object can take precedence over background objects. The model selects the public object category and skill, and the infrastructure resolves the instance. Ambiguity that the applicable rules cannot resolve remains unresolved rather than selecting an arbitrary object.

Reference admission and interaction readiness are checked separately. By default, a model may attempt a public concrete category before seeing the object; candidates requiring observation evi-

dence remain subject to their activation conditions. A valid name consequently guarantees neither existence, localization, nor readiness for manipulation. Interaction additionally requires approaching the target and satisfying affordance and access preconditions. After an object moves or drops, preserving its internal identity does not establish that the previous interaction pose remains valid.

For example, an instruction publicly specifies a drawer while other drawers are present in the scene. When the model requests navigation to `drawer`, the resolver can identify the corresponding drawer through the task mapping; a subsequent opening request using the same public name can retain the recent interaction focus. The model neither needs nor receives the drawer's simulator identifier, but must still select the opening operation and determine from returned images whether the drawer opened sufficiently. Likewise, when an instruction requires handling only dirty dishware, category resolution does not tell the model whether the dishware is dirty or replace its observation-based decision to wash it.

### H.3 Execution Results and the Model Information Boundary

After reference resolution, the executor checks interaction preconditions and executes admitted skills. Unresolved or rejected requests retain their status for internal accounting. The model-facing summary contains the requested skill and public arguments and states that the call has finished; images provide evidence of the outcome. Binding diagnostics and instance mappings follow the information-access rules in Appendix A.

System 1 provides instance-level execution support; System 2 selects actions, interprets scene observations, and assesses outcomes under the shared evaluation protocol.

## I Situated QA

Situated QA turns understanding of execution state into explicit questions and scores. Questions are generated from template semantics and reference traces, while their actual answers are resolved in the evaluated episode. We describe question types, activation, and scoring below.

### I.1 Question Types and Construction

The eight QA templates in Table 13 cover object states and spatial relations, action outcomes, active issues, new obligations, remaining requirements, and composite progress. For example, "Is the bowl now in the cabinet?" probes a relation, "What was the outcome of the placement?" probes an effect, and "Which requirements remain incomplete?" probes progress. Answers use the discrete choices associated with each question type.

All eight question types are generated in a unified pipeline from task semantics, goal predicates, and reference execution traces. Candidate questions are associated with public actions, task milestones, or other configured activation conditions and checked for uniquely resolvable answers. Selection considers diversity of question and action types and subtask coverage in composite tasks, with up to six questions per atomic episode. Scheduler replay validates activation before the selected questions are stored in a separate QA specification.

Questions use public task vocabulary without exposing private instance bindings or answer keys. Action-anchored activation matches the tool and object role, avoiding step-offset mismatches introduced by intervening answers. Answer keys are resolved from the actual evaluated episode under the snapshot rules below, rather than copied from reference traces. All eight QA types use a single selected answer from two to four choices.

The three progress-related templates use different scopes. Remaining-requirement questions ask which listed requirements remain unmet at a stopping milestone, for example, whether cleaning or storage remains incomplete. Mid-execution progress questions ask whether one or two specified requirements hold at an intermediate point, such as whether a bowl is clean and stored, with choices such as both, only the first, only the second, or neither. Composite-progress questions ask which subtask is active after a handoff, with choices identifying a child or indicating that all children are complete.

Table 13: The eight situated QA templates and the understanding they probe.

| Question type | Target of assessment |
|---|---|
| Target state | Whether an object is currently clean, open, or activated. |
| Target relation | Whether an object is inside a container or on a designated surface. |
| Post-action outcome | Whether the preceding action achieved its intended effect. |
| Active issue | Which issue or execution discrepancy remains unresolved. |
| Inserted obligation | Which additional requirement was introduced by a new instruction. |
| Remaining requirement | Which task requirement has not yet been satisfied. |
| Mid-execution progress | Which goals are complete or incomplete at the current execution point. |
| Composite progress | Child completion and handoff status within a composite episode. |

Each posted question is answered by the same task-executing model in a separate answer-only call. This call uses the currently available public execution context and performs no additional exploration or manipulation. QA therefore measures understanding of acquired information and is scored separately from physical task completion.

### I.2 Activation, Queuing, and Answer Time

Activation types include a specified tool result, a task-action step threshold, subtask handoff, a new human instruction, and a stop milestone. Each question fires at most once; child-specific questions activate only within their owning subtask. Multiple eligible questions enter a queue, and the next question is posted only when no previous question remains pending.

Action-outcome questions snapshot the evaluator state of the triggering action; delayed questions identify the original action step. State, relation, and intermediate-progress questions resolve and freeze their answer when posted, so later actions do not change their scoring reference. Resolution failures are recorded separately. Public questions contain the question, choices, and identifier; answer keys remain with the evaluator.

### I.3 Scoring

Situated QA uses the trigger rate and triggered accuracy in Appendix A.4: valid posted questions divided by valid configured questions, and correct answers divided by valid posted questions. Posted but unanswered questions count as incorrect; unposted questions are excluded from accuracy, and queued questions are not counted as posted.

Correctness uses exact matching between normalized responses and evaluator-held answer keys. Questions with unresolved answer keys are excluded as invalid and reported separately; missing or unparseable responses to valid posted questions remain incorrect. The main tables report situated QA. Because execution paths may trigger different question sets, accuracy is interpreted together with trigger rate.

## J Harness Details

SCOPER uses role-specific calls to the same fixed-parameter model for observation interpretation, agenda revision, action selection, and effect verification. Deterministic code handles structural and reference checks, memory routing, and bounded execution control. The following specification uses the same state definitions and information flow as the main text.

### J.1 Component Inputs and Outputs

Initialization produces an agenda from the task instruction without images. Observation calls receive the instruction, memory to maintain, and available images, and propose evidence-linked updates. Agenda revision receives the instruction, existing items, and committed memory, and proposes additions and modifications. The policy receives the instruction, the latest four frames, up to the latest 48 public tool-request records, routed memory, the agenda, allowed tools and argument definitions,

and recovery and anti-deadlock context, and outputs one structured tool call. Procedural guidance is supplied through item-level next intents and public tool descriptions.

Process observations from scan, navigation, and look-for calls are uniformly sampled into at most eight frames for observation maintenance; the policy uses the latest four available frames. Verification uses the existing immediate pre- and post-action frames, the request, and expected effects, without implicitly moving the camera. When QA is triggered, the same model answers in a dedicated turn using the question and the policy context, including the current instruction, images, public request history, agenda, routed memory, and control context; it outputs only the answer.

### J.2 Agenda Representation and Updates

Each agenda item stores a stable identifier, requirement, mode, status, evidence references, and optional next intent. Modes are `required`, `ongoing`, or `conditional`; statuses are `pending`, `active`, `watching`, `supported`, or `blocked`, indicating outstanding work, current focus, awaiting relevant evidence, evidence satisfying the current criterion, or a missing prerequisite. The agenda contains at most eight items with at most one active item; item order does not prescribe execution order.

Supported status is interpreted according to the requirement. A state inspection is supported when the requested information is observed; an execution requirement is supported when visual verification establishes its intended effect; an ongoing constraint is supported only for the current state and remains monitored. Discovering that a condition holds activates the associated work without establishing its fulfillment. Request history and temporary planning assumptions cannot independently support completion. Later observations contradicting supporting evidence reopen affected items.

Changes in public task requirements or relevant committed memory trigger revision rather than rewriting the agenda at every call. Prompts ask the model to propose task-scoped additions, necessary prerequisites, recovery needs, and duplicate merging; the commit layer checks fields, identifiers, references, and evidence eligibility for supported status. The model judges task relevance and semantic duplication; deterministic validation enforces the record structure and evidence requirements. The agenda advises the policy without directly defining action admissibility.

Initialization establishes explicit requirements and conditional inspection needs from the instruction. Unresolved inspection items and next intents guide the policy's choice of observation tools and targets. New public instructions are incorporated into task context; either instruction changes or relevant memory changes trigger agenda revision. QA questions are not treated as additional task requirements.

For a task to inspect a kitchen and turn off running faucets, initialization establishes an inspection need and a conditional shutoff requirement. Observing running water creates a scene fact and activates shutoff work without creating a failure incident. The shutoff requirement becomes supported only when post-action visual evidence establishes that flow has stopped. Continued flow instead records an execution incident and preserves outstanding work. Discovering one faucet does not automatically complete an inspection covering the entire kitchen.

### J.3 Memory Maintenance and Routing

Facts store an identifier, public subject, property or relation, value, confidence, frame and action references, observation step, and validity status: `current`, `stale`, or `contradicted`. These fields inform the model's semantic judgments, while deterministic routines enforce record consistency and manage memory storage and routing. Incidents store an identifier, source action, discrepancy type, evidence, creation and update steps, handling status, and recovery attempts. Obligations store an identifier, `observe` or `act` type, relevant public objects, optional source incident, creation step, and `open` or `resolved` status.

Pre-decision observation maintenance distinguishes discovery from reconciliation. Discovery adds visible objects and conditions as scene-observation facts without independently creating incidents or obligations. Reconciliation assigns candidate facts a confirmed, contradicted, unobservable, or unknown assessment. Confirmation refreshes supporting evidence; contradiction marks a fact as `contradicted`; unobservable and unknown assessments alone do not authorize negation. A

newer scene description of the same subject can supersede an earlier one. Proposals reference the relevant facts and images; the commit layer checks memory revision, reference existence, evidence eligibility, and consistency between assessments and mutations before commitment.

Default capacity is 64 facts, 16 incidents, and 16 obligations; routed context contains at most six facts, two incidents, and three obligations. Fact ordering first prioritizes currently relevant objects, then penalizes scene observations unrelated to the instruction, and then favors recent observations with stable identifiers breaking ties. Currently relevant objects come from unresolved needs and recent public interaction targets. Incidents prioritize unresolved entries and newer creation times; obligations prioritize unresolved entries and older creation times. Routing preserves validity and evidence references to distinguish historical records from current evidence.

Capacity maintenance preferentially removes stale facts and resolved records; unresolved incidents and obligations cannot be treated as handled through silent deletion. Post-action effect commitment complements pre-decision observation maintenance: the former records verification outcomes and corresponding recovery needs, while the latter handles additions and contradictions from later observations without readjudicating the same before/after evidence.

### J.4 Verification, Recovery, and Planning Assumptions

Requests record selected actions, admission determines whether they may execute, and visual verification assesses their effects. The evaluator separately determines task success under Appendix A.4; control decisions and budget exhaustion do not establish success.

Verification success indicates visual support for the intended effect, failure indicates an observable discrepancy, and ambiguity indicates insufficient evidence. Outputs include evidence and observability assessments; occlusion or a lack of obvious image change does not automatically establish failure. Success commits only supported effects; failure creates or updates incidents and associated obligations; ambiguity preserves unconfirmed effects and requests observation when needed.

Recovery state contains the original request, target, expected effect, evidence, handling phase, and remaining attempts. The controller can require observation, approach, prerequisite satisfaction, or retry. For placement recovery, `held`, `released`, and `unknown` respectively call for returning toward the destination, localization and pickup, or further observation first. Effects of recovery calls remain subject to verification.

If uncertainty persists after bounded observation or retry, recovery branches that permit continuation may retain temporary planning assumptions. These reside in recovery or control context, cannot be committed as verified facts, and cannot support agenda completion. An explicitly observed failure cannot be assumed successful; subsequent evidence may revise or revoke an assumption. Terminal branches end execution with unresolved state retained.

### J.5 Anti-Deadlock Control and Execution Order

The exploration ledger groups scans, searches, and navigation by room or semantic target and tracks observation and preparation calls without advancing interaction. At the corresponding limits, the controller rejects repetitive candidates before execution, supplies a reselection reason, and restricts tools, promoting approach and then interaction where appropriate. Relevant new evidence or admitted physical calls reset the corresponding counters under the control rules; precondition-rejected calls and changes in timestamps or step indices do not.

Recovery observations, retries of the original action, and model reselection each have finite budgets. Controller-required recovery calls take precedence over ordinary action selection, with the agenda serving as policy context. QA uses dedicated answer turns outside exploration candidate selection. Execution, answer, and stop requests each consume a high-level step; internal model calls and reselection do not add environment steps.

**Default control limits.** Within a no-progress phase, the default limits are two scans, two searches and two navigation calls per semantic target, four observation calls in total, and six observation/navigation preparation calls in total. A rejected candidate permits at most two additional model reselection requests. Recovery contexts default to two retry rounds, while dedicated placement re-

covery additionally permits at most eight recovery actions. These limits apply within their respective ledgers and do not extend the episode's environment-step budget.

Exhausted dedicated placement recovery terminates with a control failure. Generic recovery branches that permit continuation retain unconfirmed state and separately marked planning assumptions. Later evidence revises these assumptions.

Algorithm 1 orders observation maintenance, agenda revision, routing, action selection, and effect commitment; Algorithm 2 details bounded action selection. $M$ denotes memory, $A$ the agenda, $R$ recovery state, $L$ the control ledger, $H$ public request history, and $W$ the available image buffer. Calls parameterized by $\theta$ share the same model; other operations perform deterministic control or environment interaction.

Observe comprises discovery and reconciliation with their respective update permissions. RECENTREQUESTS retains up to 48 public requests; policy and QA calls share context $\kappa$. UpdateRecovery maintains phases, budgets, and permitted planning assumptions. Agenda acceptance checks structure, references, and evidence eligibility, while the model handles task scope and semantic merging.

$X$ contains the initial instruction and subsequent public task messages, while $E$ tracks processed image identifiers. Task updates exclude QA questions and can trigger agenda revision without waiting for a new visual fact.

Updated recovery state and control ledgers are returned to the main loop.

Memory maintenance tracks frame identifiers and update responsibilities. A post-action image can support discovery of other scene information, while its action-effect assessment is committed through verification. Repeated frames retain their existing evidence identity.

## K EXECUTION TRACE VISUALIZATION

The simulated execution traces in Figure 4 and this section are produced by Qwen3.8-27B with SCOPER-harness. Some redundant intermediate exploration steps and frames are omitted for readability. Real-data examples illustrate next-tool-call evaluation conditioned on reference execution prefixes.

**Execution failures and disturbances.** Figure 5 shows a misplaced plant pot being located, retrieved, and placed correctly. Figure 6 illustrates retrieval after a slipped grasp, while Figure 7 shows visual search and corrected navigation after reaching the wrong target. Figure 8 illustrates inspection and retry after a partial drawer opening. Figures 9 and 10 follow a two-object storage task: after storing the first object, the cabinet closes again and must be reopened before storing the second.

**Observation-dependent requirements and action dependencies.** Figures 11 and 12 show discovery of a dirty mug followed by cleaning and storage. Figure 13 illustrates a safety inspection in which observations identify a running faucet and an open fridge. Figures 14 and 15 show observation of a dirty bowl activating the instruction-specified cleaning and storage requirement during a tomato-placement task. Figure 16 illustrates sequential dependencies: the knife must be put down after slicing before the apple slices can be picked up and placed.

**Real-observation decision nodes.** Figure 17 illustrates movement and reinspection revealing evidence relevant to the next action, such as a fallen cushion or remaining litter. Figure 18 illustrates synthetic early-stop decision nodes: picking, placing, opening, or approaching is still incomplete at the marked endpoint. These are imposed truncations of reference demonstrations, rather than recorded execution malfunctions. Later reference frames illustrate eventual outcomes and are excluded from model inputs.

**Algorithm 1** SCOPER execution with evidence memory and an agenda

**Require:** Instruction $x$, initial image $o$, tools $\mathcal{S}$, step budget $B$

$M, R, L, H \leftarrow \emptyset$; $W \leftarrow [o]$; $b \leftarrow B$; $z \leftarrow$ ACTIVE
$X \leftarrow x$; $E \leftarrow \emptyset$
$A \leftarrow$ ACCEPTINITIAL($\text{Initialize}_\theta(X)$)
**while** the episode is not terminated and $b > 0$ **do**
  $X \leftarrow$ INCORPORATEPUBLICTASKUPDATES($X$)
  $J \leftarrow$ UNPROCESSEDEVIDENCE($W, E$)
  **if** $J \neq \emptyset$ **then**
    $\Delta_M \leftarrow \text{Observe}_\theta(X, M, J)$
    $M \leftarrow$ COMMIT($M, \Delta_M$)
    $E \leftarrow E \cup$ EVIDENCEIDENTIFIERS($J$)
  **end if**
  **if** task context or relevant committed memory has changed **then**
    $\Delta_A \leftarrow \text{ReviseAgenda}_\theta(X, A, M)$
    $A \leftarrow$ ACCEPTAGENDA($A, \Delta_A, M$)
  **end if**
  $\widetilde{M} \leftarrow$ ROUTE($M, X, H, R$); $V \leftarrow$ LATESTFOUR($W$)
  $c \leftarrow$ CONTROLCONTEXT($\mathcal{S}, R, L$)
  $\kappa \leftarrow (X, V,$ RECENTREQUESTS($H$)$, A, \widetilde{M}, c)$
  **if** a question $q$ is pending **then**
    $a \leftarrow \text{Answer}_\theta(q, \kappa)$
    SUBMITANSWER($a$); $b \leftarrow b - 1$
    $H \leftarrow H \parallel [$PUBLICREQUEST($a$)$]$; **continue**
  **end if**
  **if** $z =$ ANSWERONLY **then**
    **break**
  **end if**
  $(a, R, L) \leftarrow$ GUARDEDSELECT($\theta, \kappa, R, L$)
  **if** $a =$ EXHAUSTED **then**
    **terminate** with control-exhausted status
  **end if**
  $H \leftarrow H \parallel [$PUBLICREQUEST($a$)$]$
  **if** $a =$ `stop` **then**
    STOPANDFREEZESCORE; $b \leftarrow b - 1$; $z \leftarrow$ ANSWERONLY
    **continue**
  **end if**
  $o^- \leftarrow$ CURRENTIMAGE($W$)
  $(o^+, \mathcal{V}, e) \leftarrow$ BINDANDEXECUTE($a$); $b \leftarrow b - 1$
  **if** the admitted action requires effect verification **then**
    $g \leftarrow$ EXPECTEDEFFECT($a$)
    $v \leftarrow \text{Verify}_\theta(a, g, o^-, o^+)$
    $M \leftarrow$ COMMITEFFECT($M, a, v$)
    $R \leftarrow$ UPDATERECOVERY($R, a, v, M$)
  **end if**
  $L \leftarrow$ UPDATELEDGER($L, a, e, M$)
  $W \leftarrow$ APPENDIMAGES($W, \mathcal{V}, o^+$)
**end while**

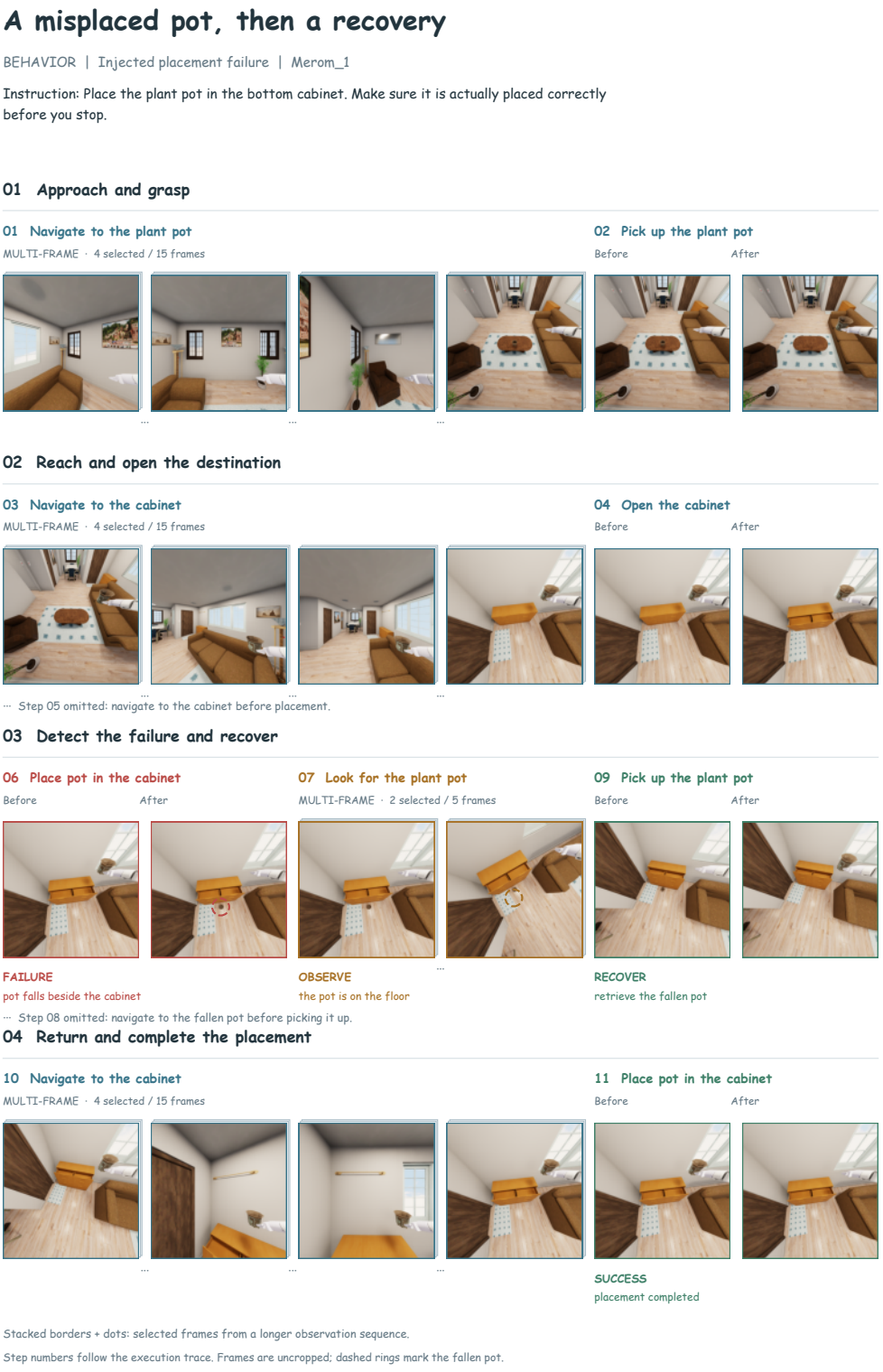


Figure 5: Sample 1: Retrieving a fallen plant pot and completing placement.

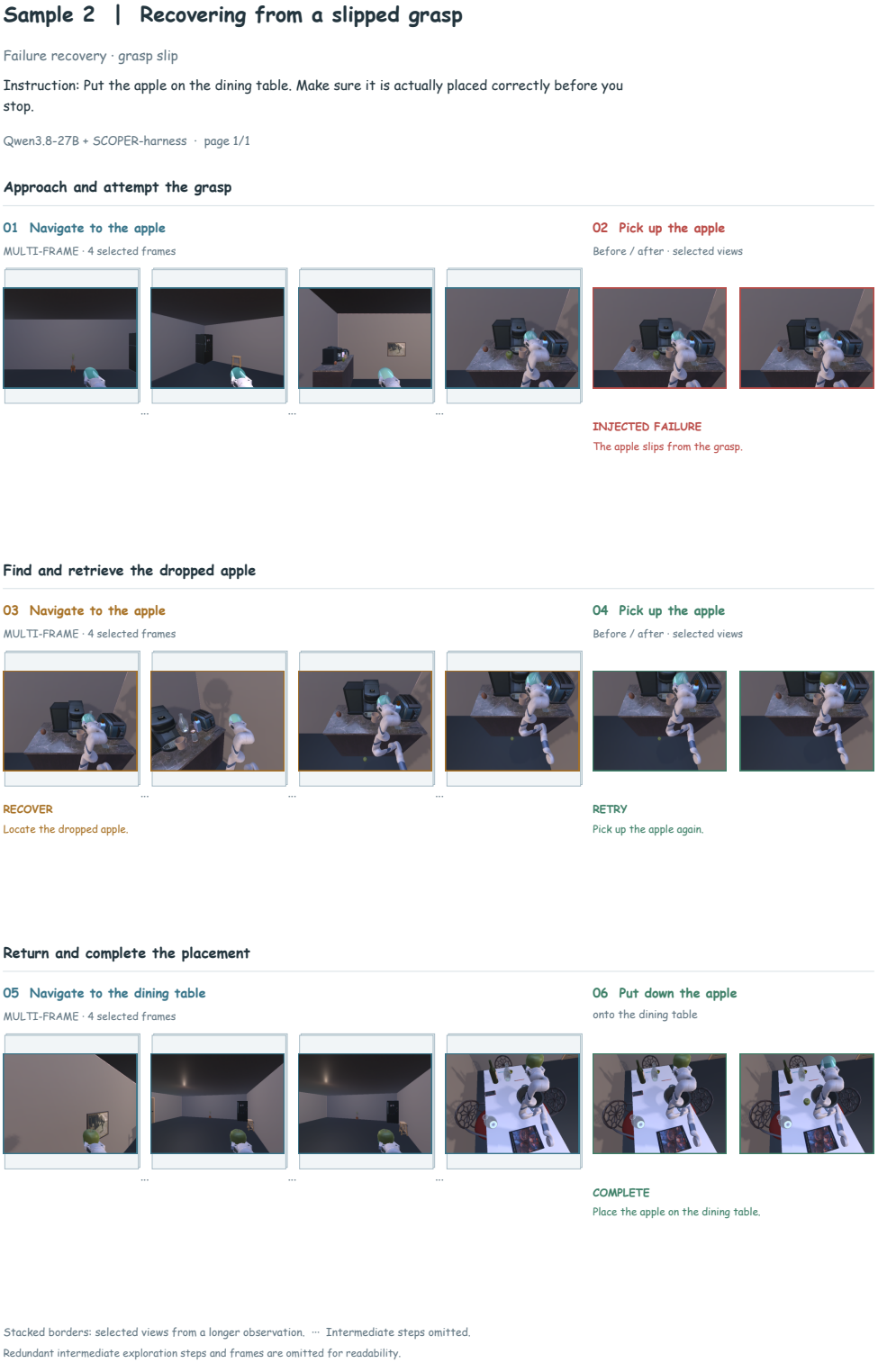


Figure 6: Sample 2: Retrieving an apple after a slipped grasp.

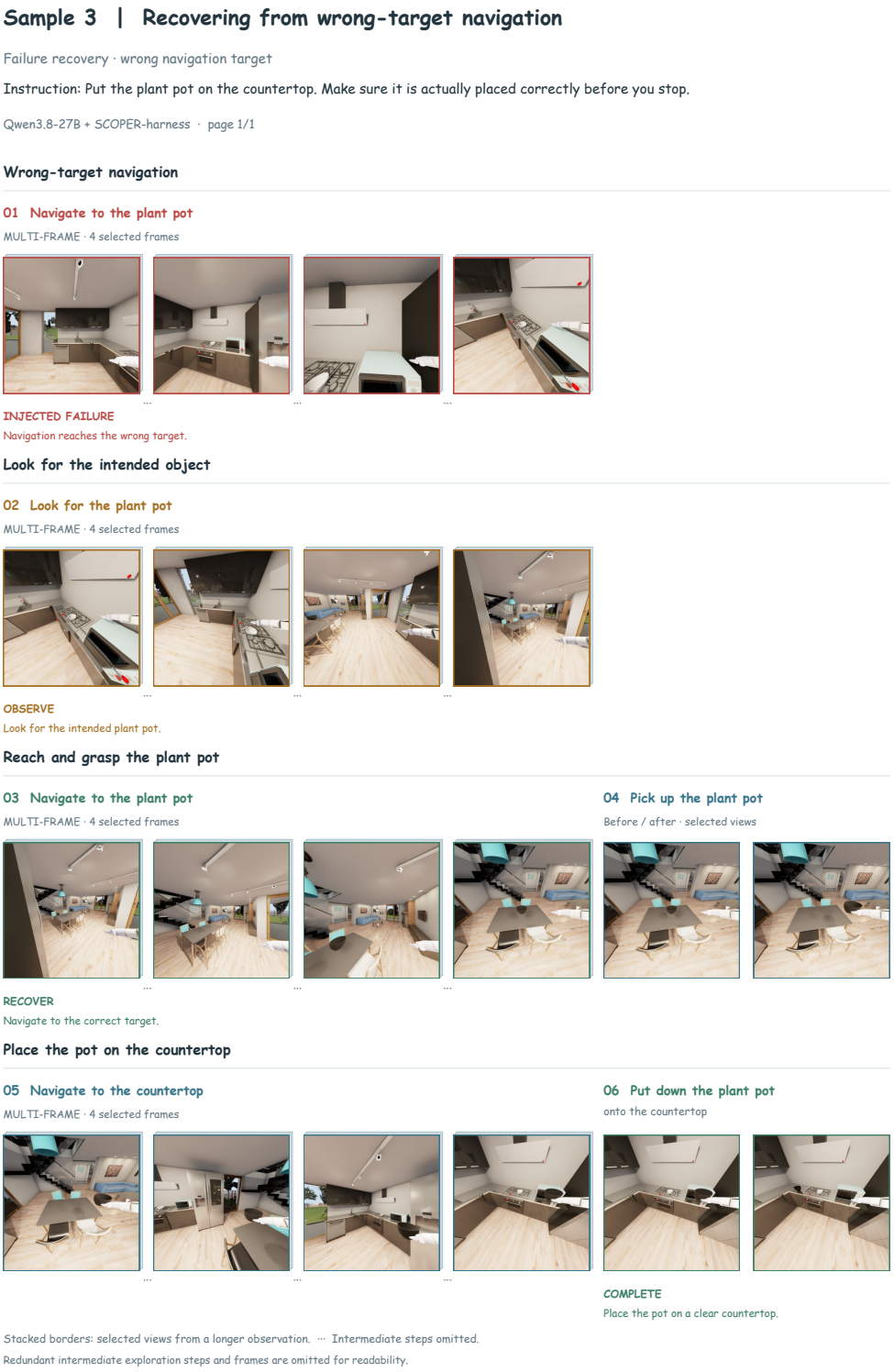


Figure 7: Sample 3: Correcting navigation to the wrong target.

**Algorithm 2** Bounded action selection

**Require:** Model $\theta$, policy context $\kappa$, recovery state $R$, ledger $L$

**if** a required recovery call is due and its budget permits **then**
  **return** (required call, $R, L$)
**end if**
Apply recovery-exhaustion rules to $R$ when applicable
**if** these rules terminate execution **then**
  **return** (EXHAUSTED, $R, L$)
**end if**
**for** a bounded number of policy requests **do**
  $\kappa \leftarrow \text{REFRESHCONTROLCONTEXT}(\kappa, R, L)$
  $a \leftarrow \pi_\theta(\kappa)$
  **if** $\text{ADMISSIBLE}(a, R, L)$ **then**
    **return** $(a, R, L)$
  **end if**
  $L \leftarrow \text{ADVANCECOMMITMENT}(L, a)$
  $\kappa \leftarrow \text{ADDRESELECTIONREASON}(\kappa, a, L)$
**end for**
**return** (EXHAUSTED, $R, L$)

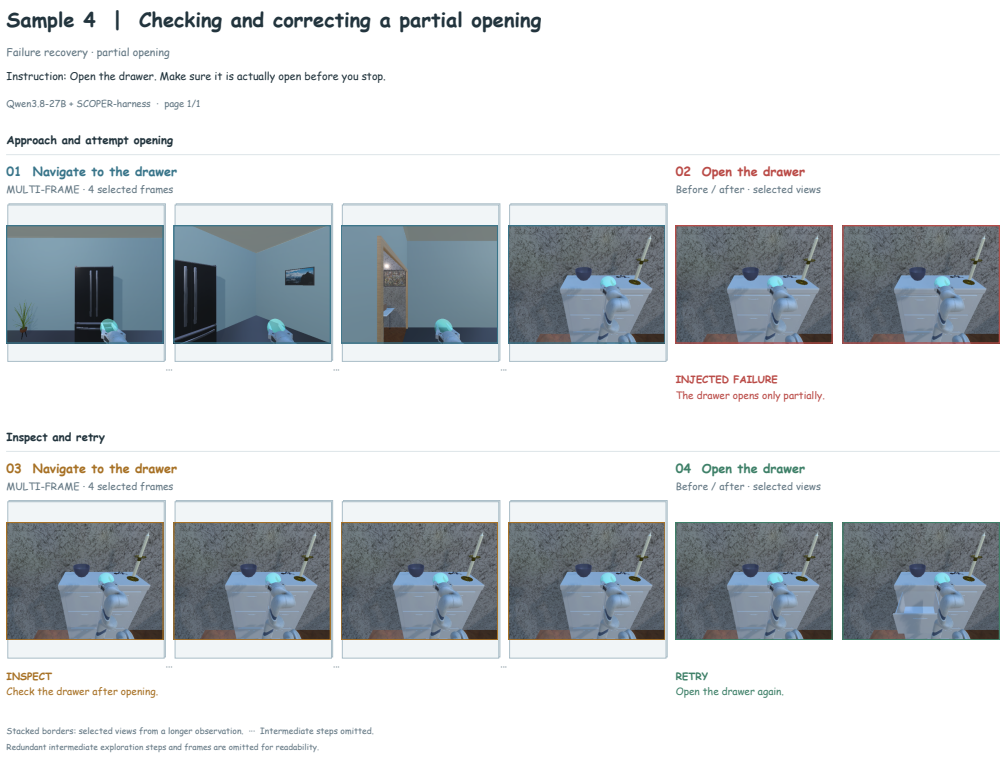


Figure 8: Sample 4: Inspecting and retrying a partial drawer opening.

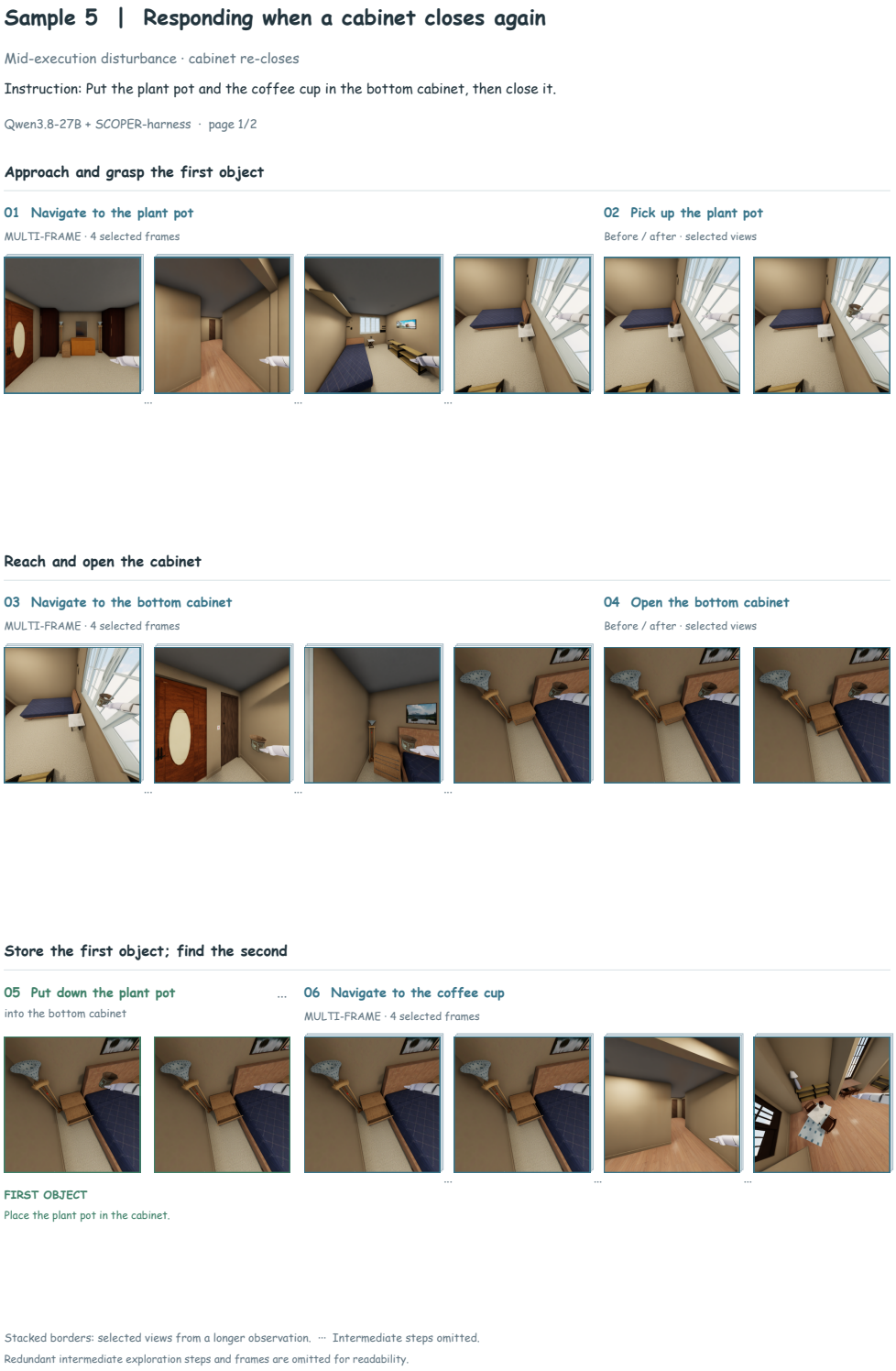


Figure 9: Sample 5, part 1: Storing the plant pot and approaching the coffee cup.

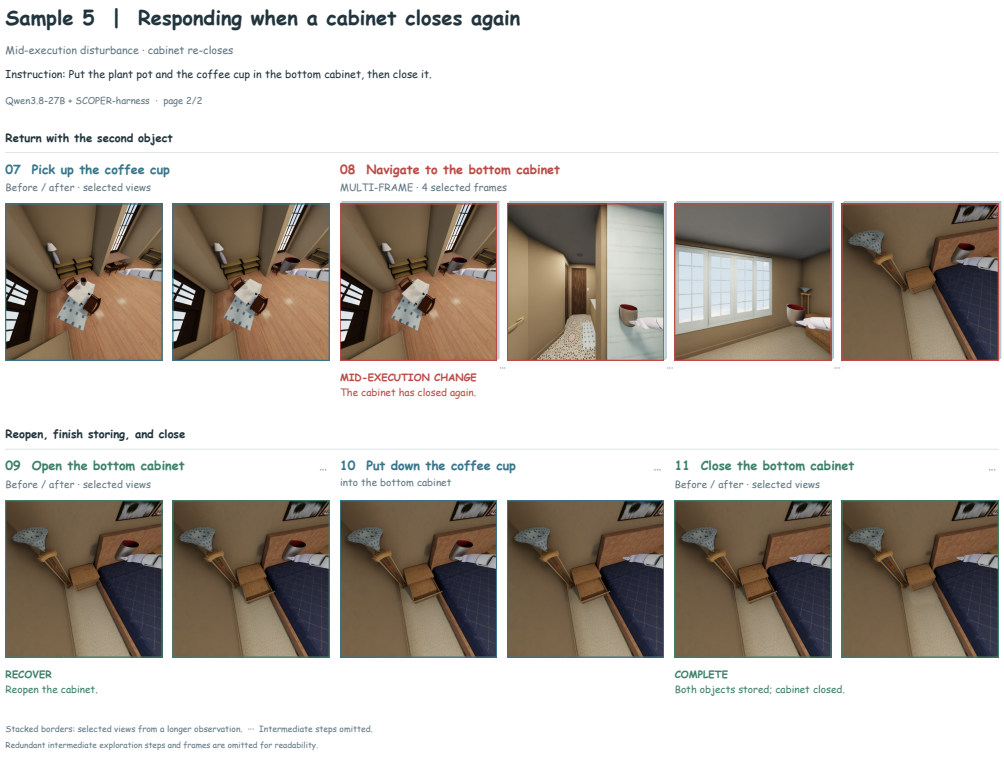


Figure 10: Sample 5, part 2: Reopening the cabinet to store the coffee cup.

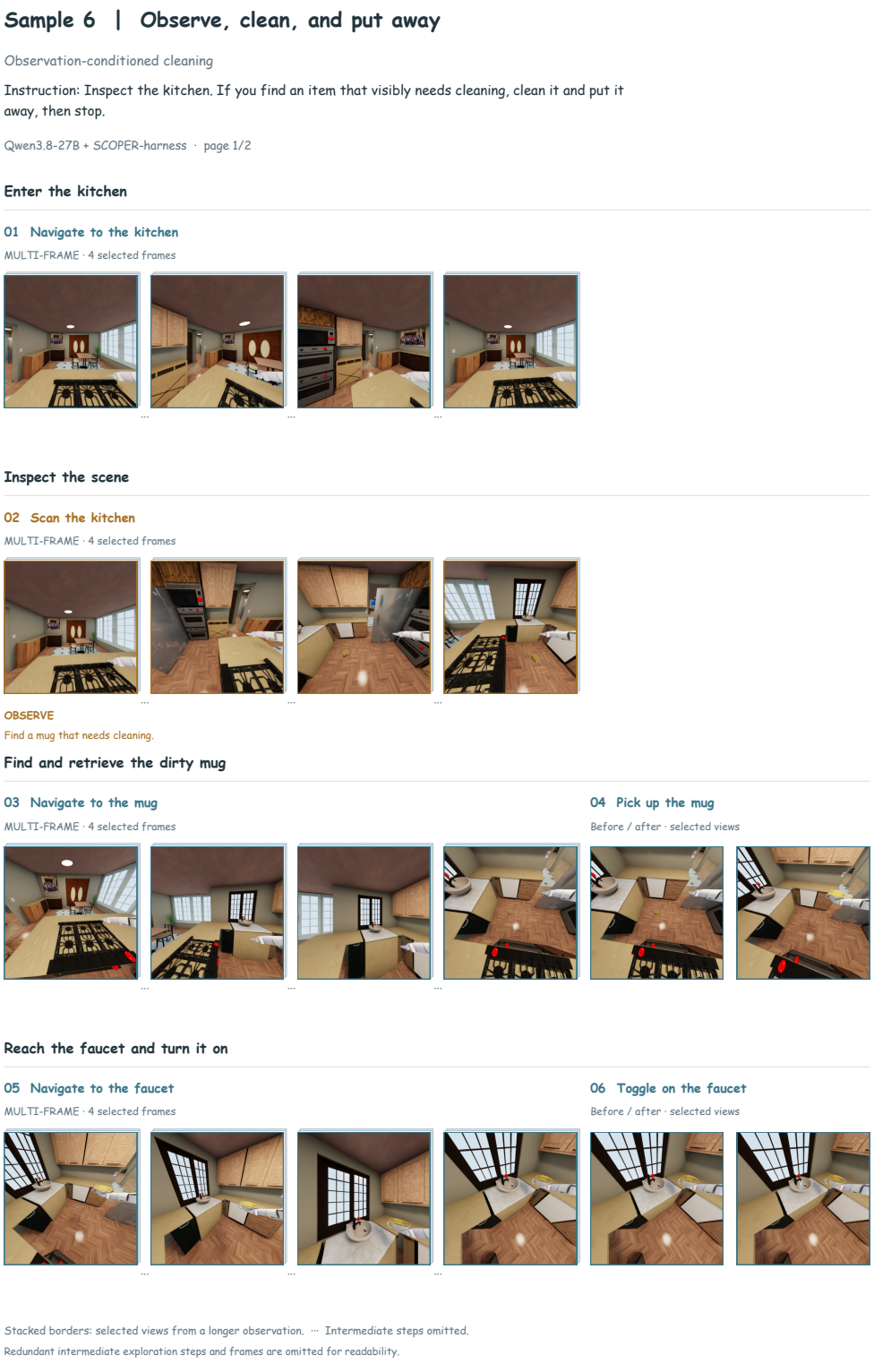


Figure 11: Sample 6, part 1: Finding a dirty mug and bringing it to the faucet.

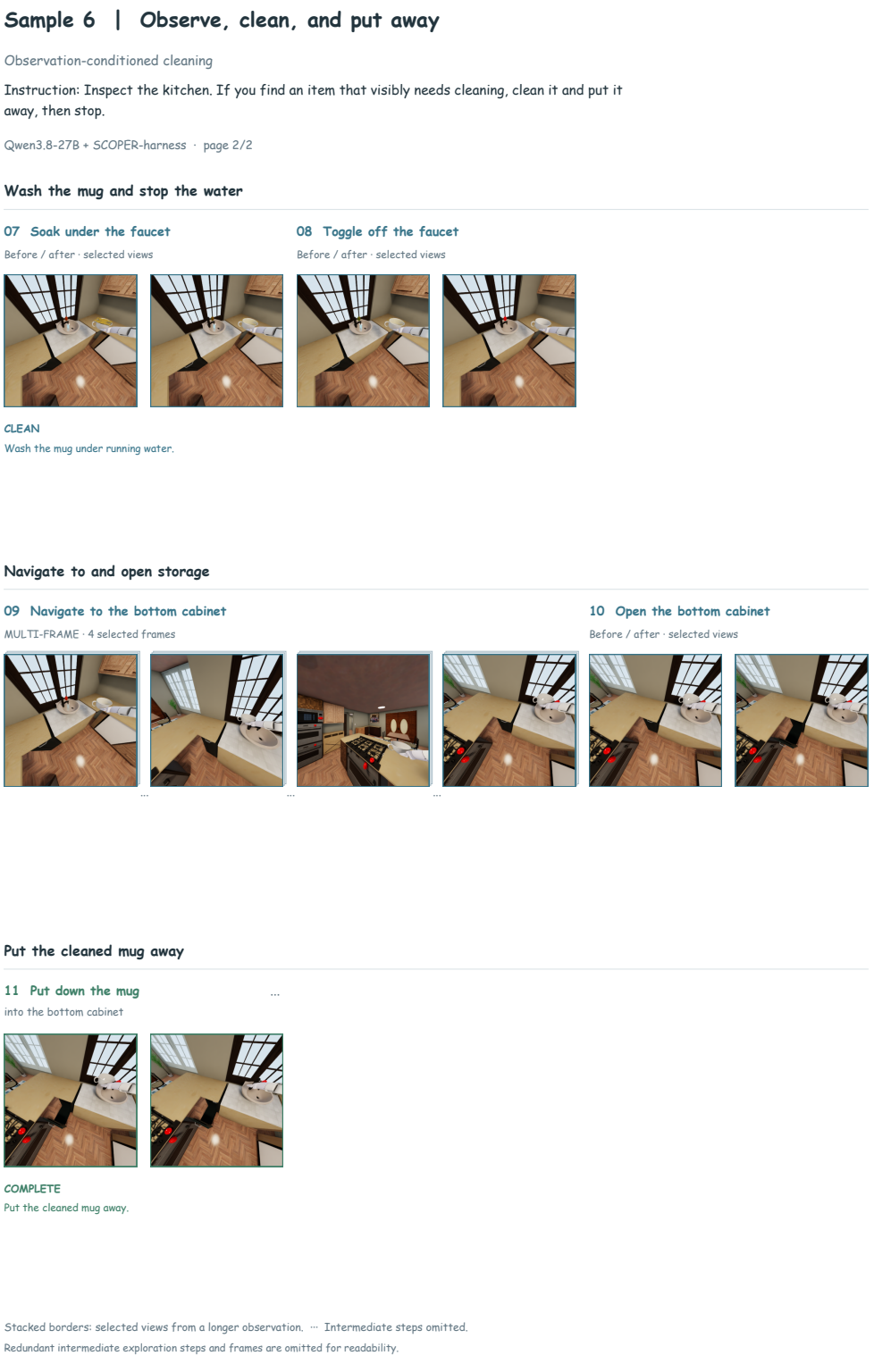


Figure 12: Sample 6, part 2: Washing the mug and putting it away.

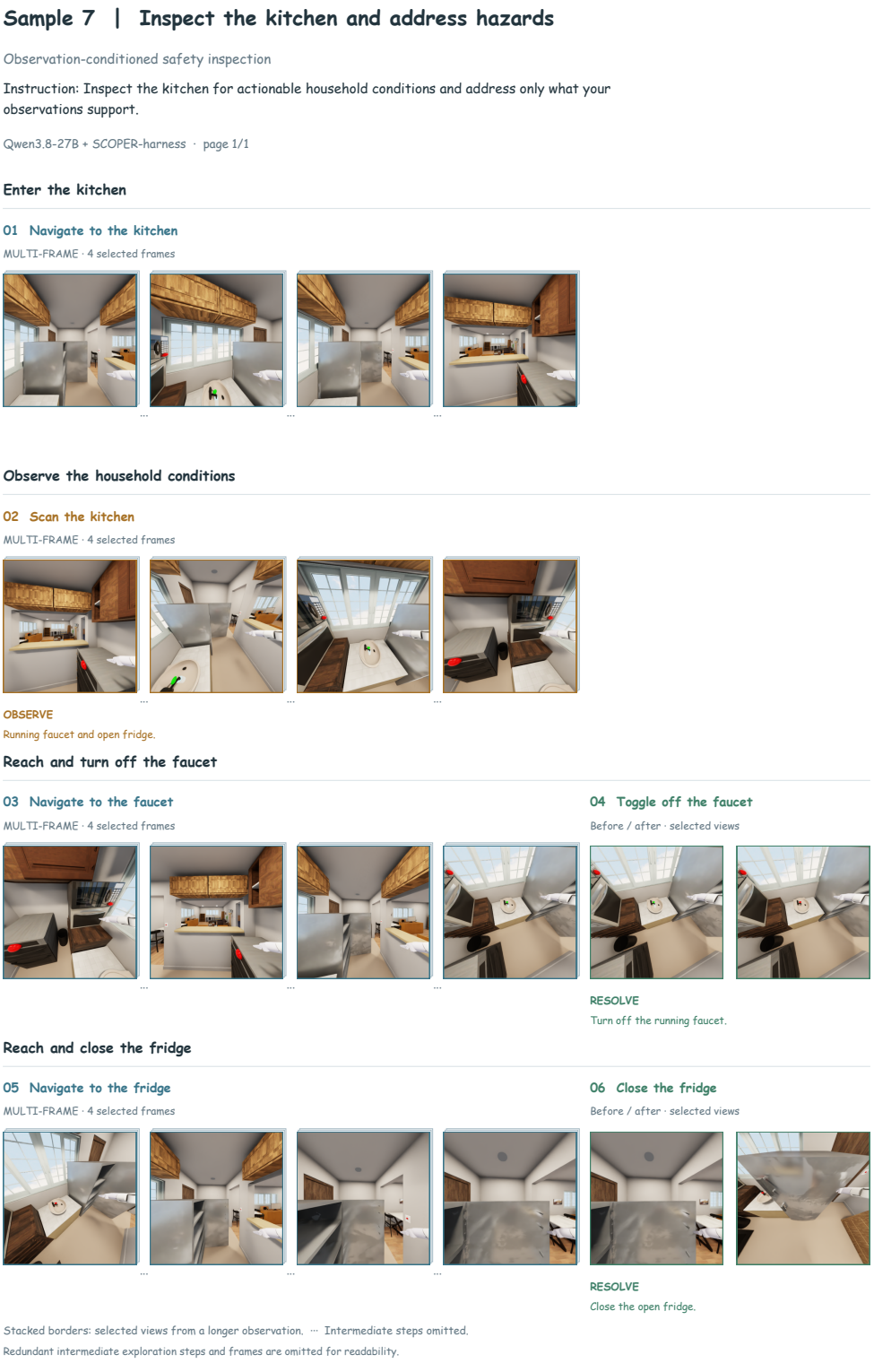


Figure 13: Sample 7: Turning off an observed running faucet and closing an open fridge.

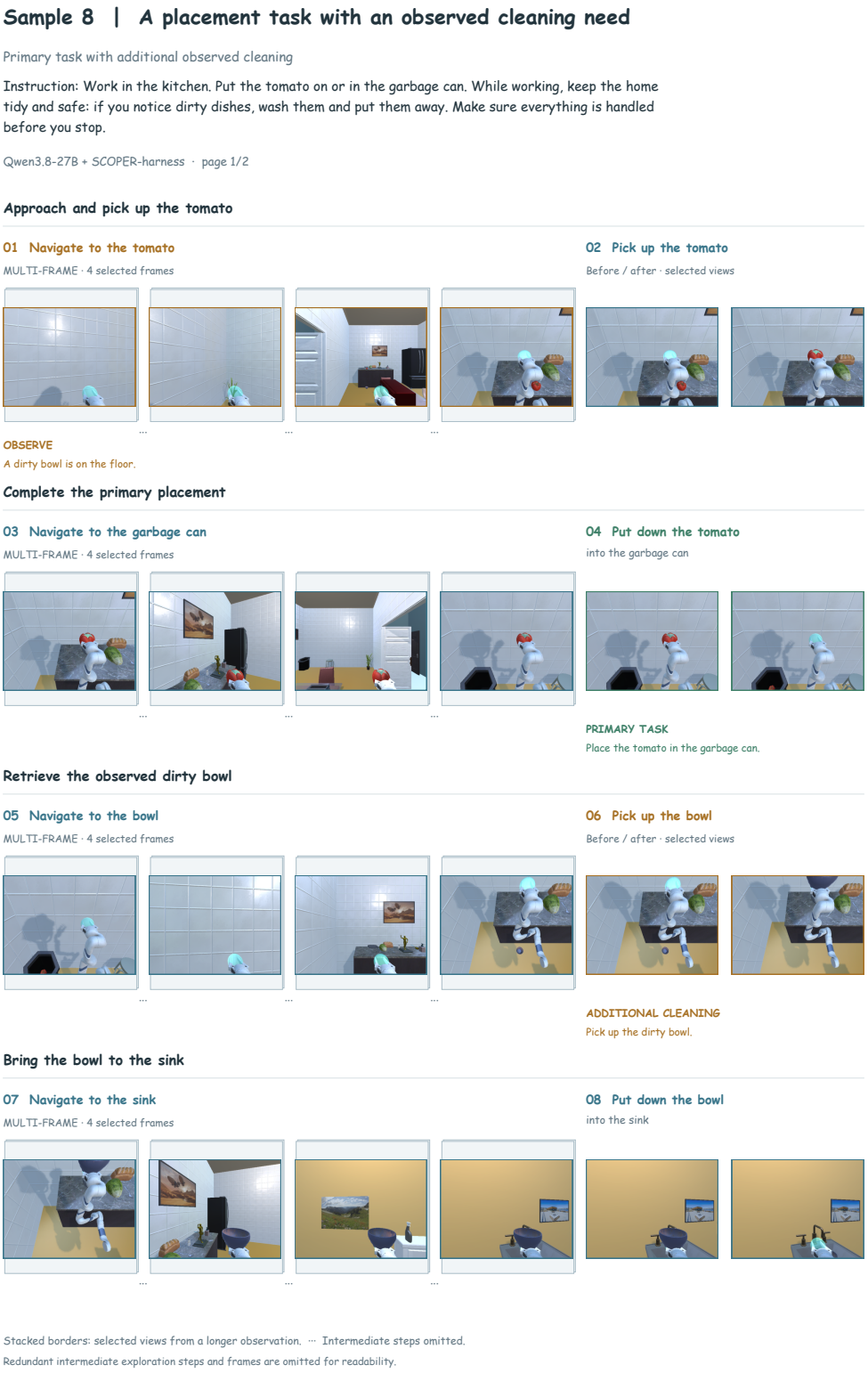


Figure 14: Sample 8, part 1: Placing the tomato and retrieving an observed dirty bowl.

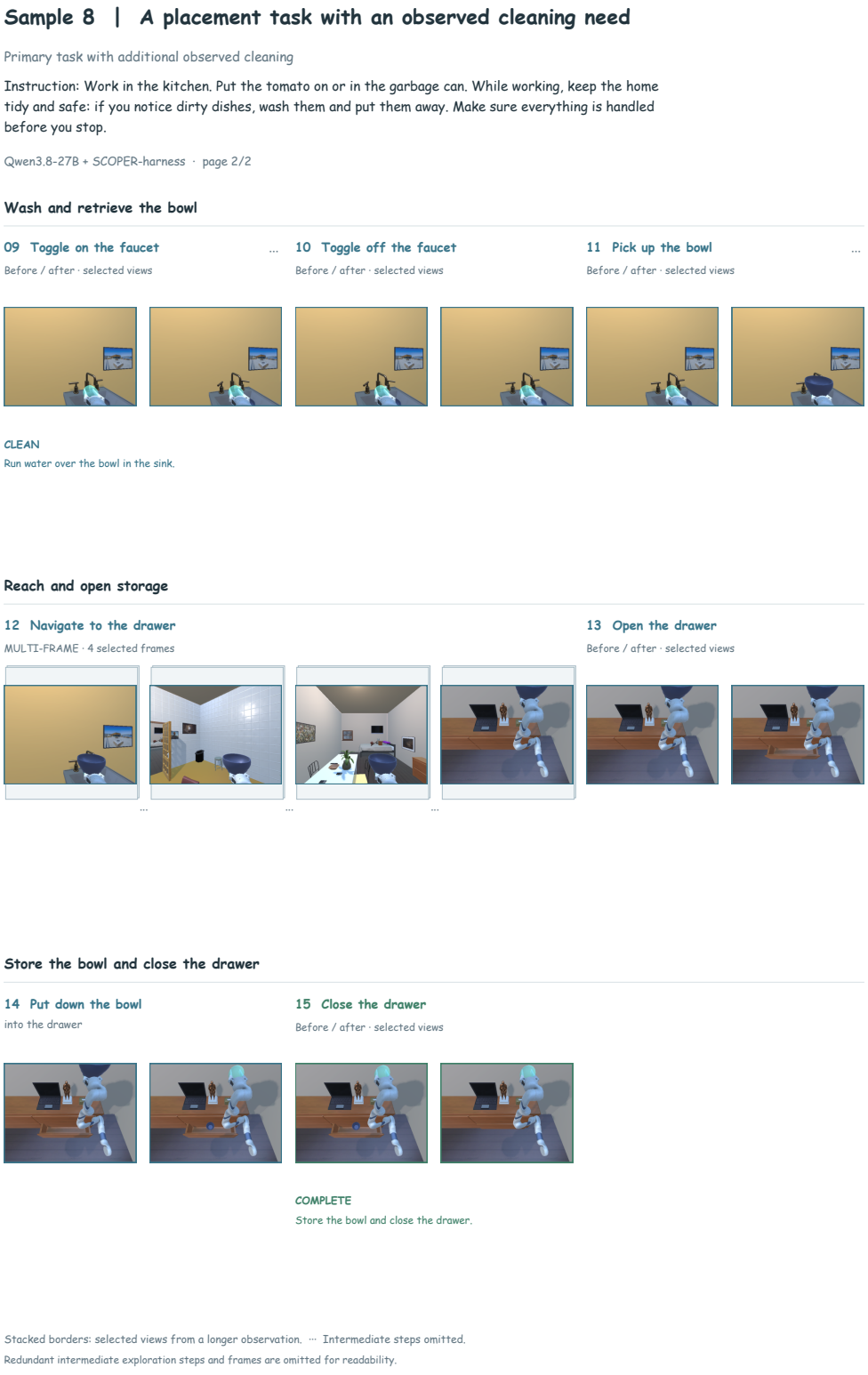


Figure 15: Sample 8, part 2: Washing and storing the bowl as required by the instruction.

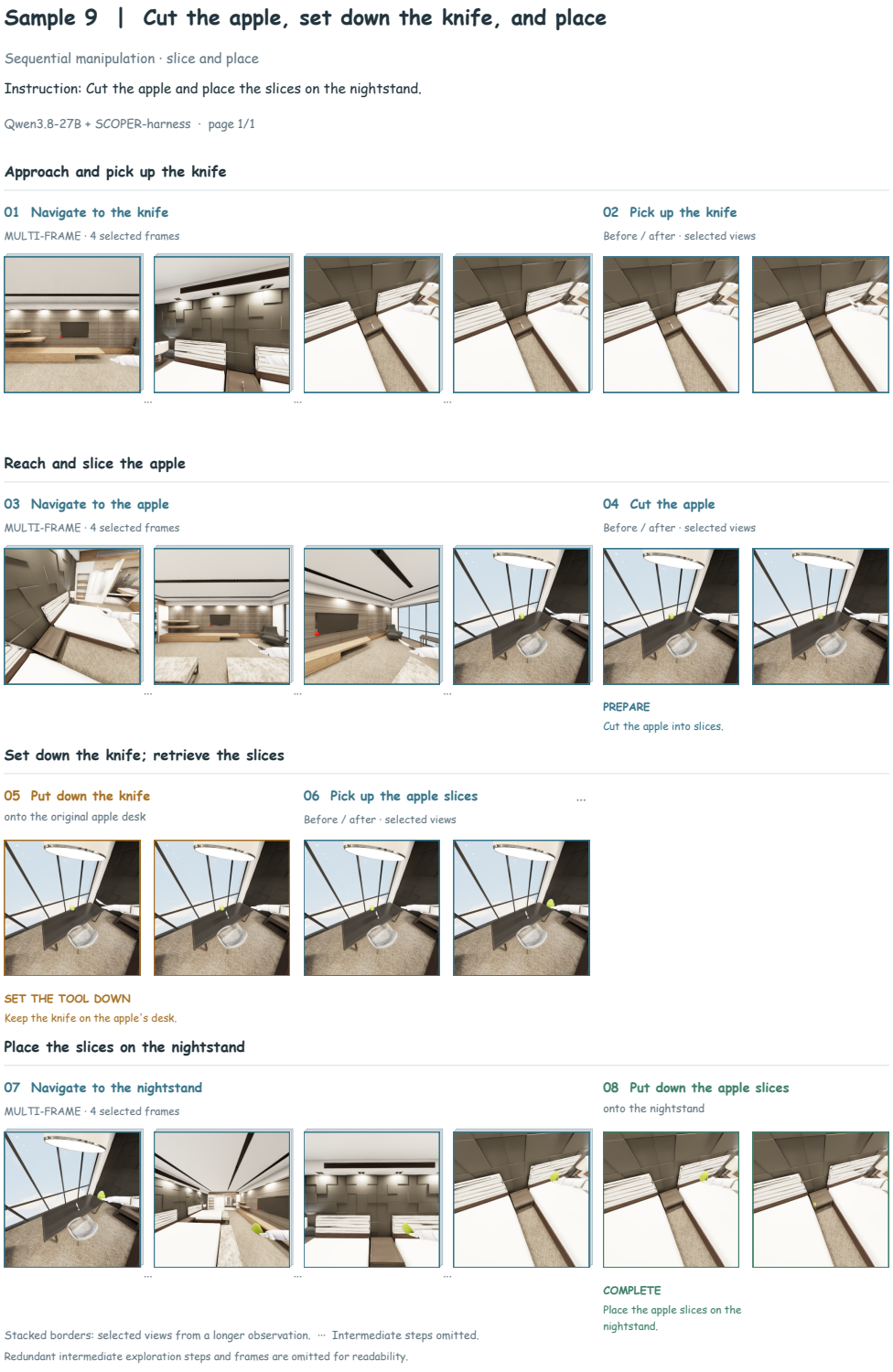


Figure 16: Sample 9: Setting down the knife before picking up and placing the apple slices.

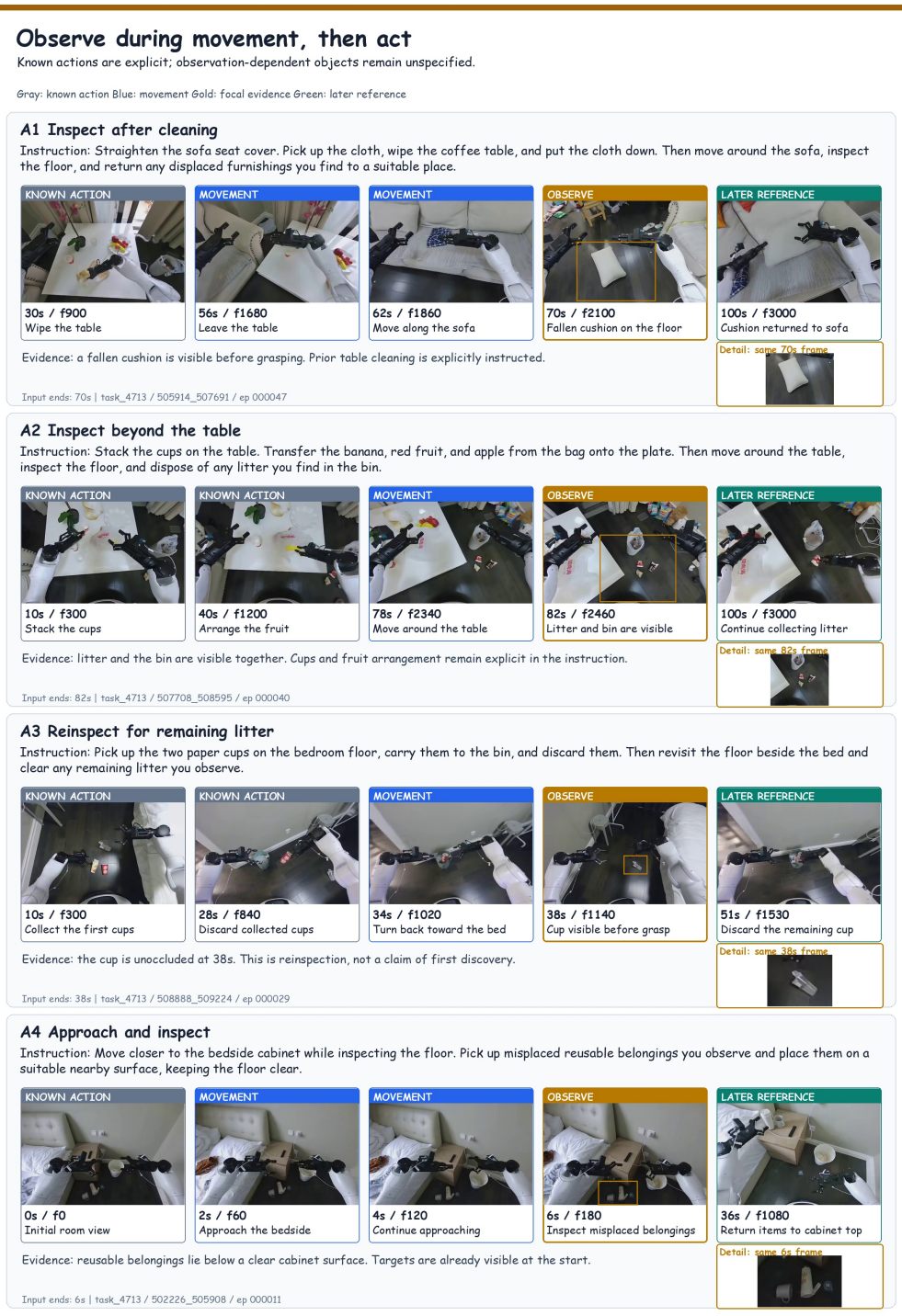


Figure 17: Real-data examples: Observing action-relevant evidence through movement and reinspection.

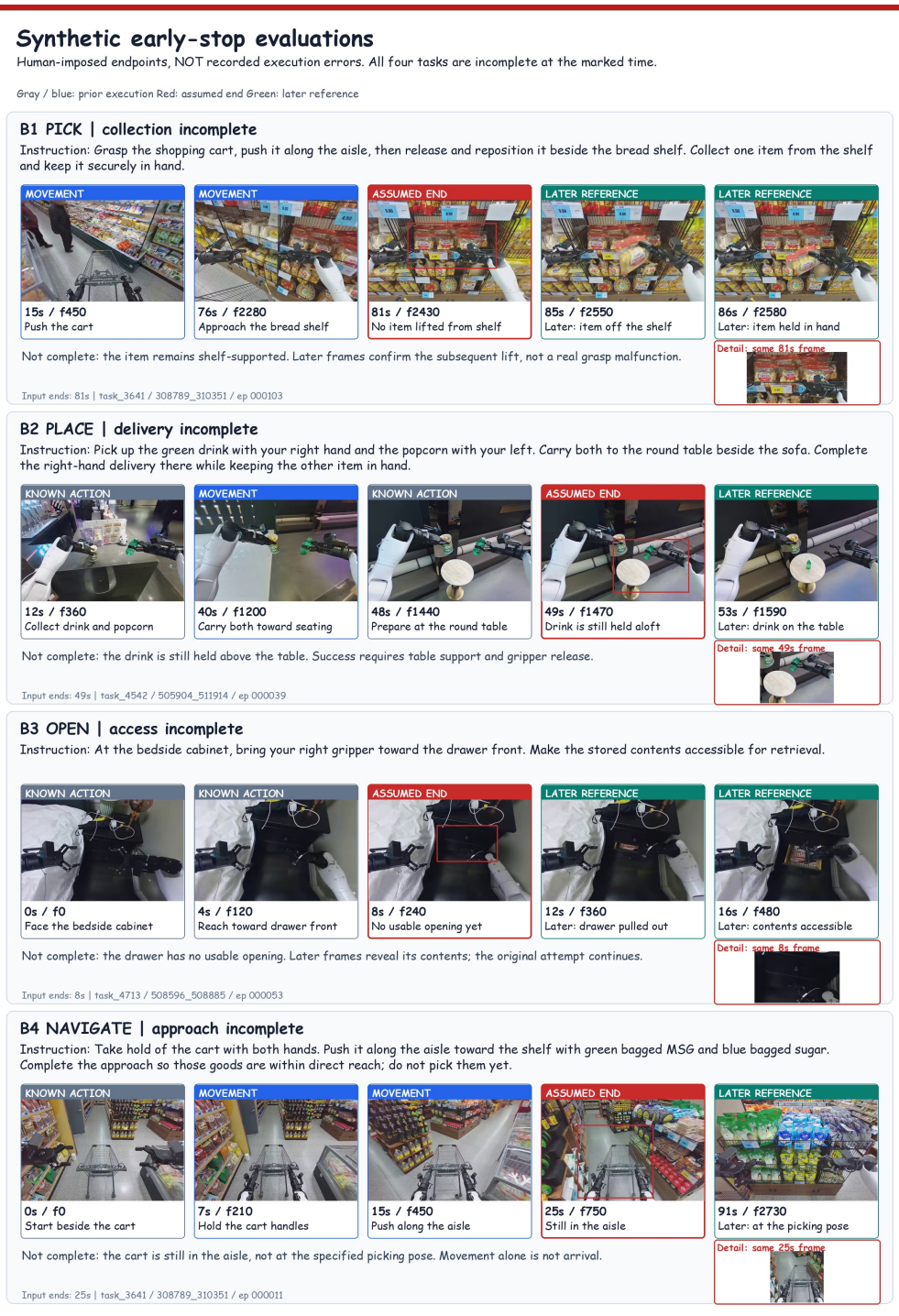


Figure 18: Real-data examples: Synthetic early-stop nodes with incomplete actions.

## L HARNESS MODEL-CALL OVERHEAD

Direct execution uses one model call per high-level step. SCOPER adds role-specific calls around action selection: the policy usually makes one call per step; discovery makes one call when new images are available; reconciliation makes one call when existing facts can be checked; agenda revision makes one call when the instruction or committed memory changes; and the verifier makes one call after an admitted operation requiring outcome verification. These calls support action selection, acquisition of new scene facts, assessment of earlier evidence, task-progress updates, and visual outcome checks, respectively. Averaged over evaluation on the complete 870-task benchmark, SCOPER uses approximately $3.7\times$ as many model calls as direct execution. Internal calls do not add environment-interaction steps; both methods retain the same high-level step budgets.