RECALL & ROLL: BENCHMARKING HOW EMBODIED AGENTS REMEMBER, ACT, AND UPDATE MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Embodied agents must recall past experience and track progress while acting. We introduce Recall & Roll Benchmark (R&R-Bench) to evaluate these requirements across navigation and manipulation. It comprises 300 contexts across 228 simulated houses, totaling 43 hours of experience and 1,772 queries. Four task families, Finding, Restore, Resume, and Routine, cover distinct memory requirements, with variations in request scope, evidence source, and target concealment. Three evaluation conditions examine memory QA, execution given correct goals, and the full pipeline, supporting analysis of retrospective recall and its relationship to task execution. Across the evaluated systems, visual memory better supports spatial recall and passively observed information, while textual memory better supports unfinished-work recovery, storage-rule inference, and progress tracking. Guided by these findings, we introduce R&R AGENT as a reference system that combines textual records, tagged visual evidence, and request-conditioned retrieval. It achieves the highest task-average memory-QA accuracy and full-pipeline success across all four families among the evaluated methods that construct memory from experience. Separate placement experiments examine retrieved visual goals for precise manipulation. R&R-Bench provides a common testbed for characterizing embodied memory requirements, identifying limitations of existing approaches, and informing future agent design.



Figure 1: Overview of R&R-Bench: 1,772 queries, 300 long-horizon contexts, and 228 houses. Left: Queries grouped by task family (inner ring) and Single/Multi request scope (outer ring). Middle: Query-weighted context-duration distributions. Dots and thick lines mark medians and interquartile ranges. Right: Six examples from the 228 houses, illustrating varied room layouts and furnishings, scaled independently for visibility.

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.

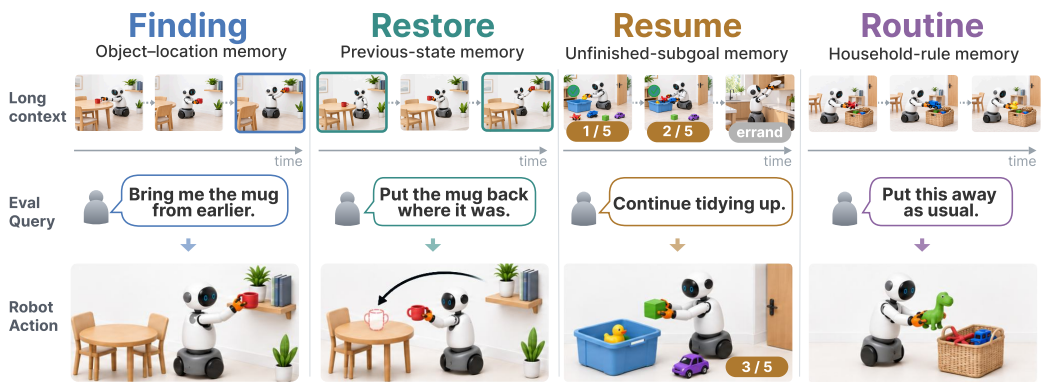

Figure 2: Overview of R&R-Bench. Long-context experience supports later memory-grounded actions across four task families: Finding recalls object locations, Restore recovers previous states, Resume tracks unfinished subgoals, and Routine retrieves household rules. Illustrative examples.

1 INTRODUCTION

Many embodied tasks require agents to remember past experience and track ongoing progress (Datta et al., 2022; Huang et al., 2023). For example, consider a household robot asked to "*put the cups back where they were*." The robot must recall which cups were moved, where they are now, and where each originally belonged. As the robot puts each cup back, it must track which cups have been returned and which still need to be moved. The current observation alone cannot reveal their earlier locations, and an initial plan does not record the outcomes of subsequent actions. Fulfilling such requests therefore requires both recalling past experience and updating memory while acting.

These demands involve two memory functions. *Retrospective memory* (Smith et al., 2000) recovers information from experience before a request, while *execution state memory* (Huang et al., 2023) maintains information acquired while carrying it out. Their roles depend on the request. Finding an object requires its location, restoring it requires an earlier state (Weihs et al., 2021), resuming work requires past completion records, and following a routine requires inferring rules from previous examples (Wu et al., 2023). Existing benchmarks study memory in manipulation, object search, and reasoning over interaction histories (Dai et al., 2026; Sun et al., 2026; Chen et al., 2025; Zhang et al., 2026), but offer limited coverage of these varied requirements across navigation and manipulation. Recovering task-relevant information from memory does not guarantee successful task execution, while task success alone does not reveal where difficulties arise (Sun et al., 2026). Our goal is to understand what agents need to remember for different tasks and how they use and update that memory to complete them.

To investigate this question, we introduce Recall & Roll Benchmark (R&R-Bench). It comprises 300 recorded contexts across 228 simulated houses, totaling 43 hours of experience and 1,772 queries (Figure 1). Four task families, Finding, Restore, Resume, and Routine, cover the information requirements above, with further breakdowns by direct interaction or passive observation, request scope, and target concealment. Memory QA evaluates retrospective memory through memory construction, retrieval, and reasoning. Execution with ground truth instructions probes execution-state memory within the full planning-and-control loop, while removing errors in retrospective goal recall. Execution with recalled instructions evaluates the full pipeline, combining retrospective and execution-state memory to complete the task. Together, these evaluations support fine-grained analysis of memory construction and use, progress tracking, and action execution within a VLA system.

Our primary contribution is R&R-Bench and the empirical characterization of embodied memory that it enables. Our comparisons show that the usefulness of retained evidence depends on the request: visual memory supports recall of passively observed objects, while explicit textual records help recover unfinished work. To explore how these findings can inform agent design, we introduce R&R AGENT as a reference system within the shared two-stage VLA framework. It preserves textual records and tagged visual evidence for request-conditioned retrieval and updates memory during execution. Its improvements illustrate the practical value of combining these mechanisms under the evaluated conditions. A separate placement study investigates retrieved visual goals for conveying spatial details to the low-level policy.

Table 1: Benchmark comparison. Low-level manipulation predicts arm and gripper commands. Memory requirements: S: object locations and scene states; E: events, their order, and task progress; R: inferring rules from experience and applying them to new cases.

Benchmark	Long context (> 5 min)	Navigation + manipulation	Low-level manipulation	Online memory	Multiple evaluation modes	Memory requirements
MemoryBench (Fang et al., 2025)	✗	✗	✓	✓	✗	S / E
RoboMME (Dai et al., 2026)	✗	✗	✓	✓	✗	S / E
RoboMemArena (Lei et al., 2026)	✗	✗	✓	✓	✗	S / E
FindingDory (Yadav et al., 2026)	✓	✗	✗	✗	✓	S / E
STARBench (Chen et al., 2025)	✓	✓	✗	✓	✗	S / E
WorldLines (Zhang et al., 2026)	✓	✗	✗	✗	✓	S / E
LongAct (Zhu et al., 2026)	✓	✓	✗	✓	✗	S / E
R&R-Bench (ours)	✓	✓	✓	✓	✓	S / E / R

Our contributions are threefold.

- We introduce R&R-Bench, a benchmark with validated experience contexts in multi-room household environments. Four task families, diagnostic task variations, and three evaluation conditions support the study of retrospective memory and memory use during embodied execution.
- We characterize the strengths and limitations of evaluated memory strategies across task requirements and memory readers. The results reveal differences in the usefulness of textual and visual evidence and identify execution challenges that persist even when correct goals are available.
- We provide R&R AGENT as a reference system informed by these findings. Its evaluation illustrates how the benchmark can guide the integration of memory preservation, retrieval, and online updating, while a separate placement study explores the use of retrieved visual goals.

2 RELATED WORK

Memory-augmented VLM/VLA methods. Recent advances in VLAs have motivated growing interest in longer-horizon tasks, driving the development of policies that retain and use information across extended interactions. MemoryVLA (Shi et al., 2026), MemoAct (Tan et al., 2026), and HAMLET (Koo et al., 2026) incorporate memory into action prediction, while LingBot-VA (Li et al., 2026) models video and action sequences jointly. Longer histories require selective retention and reasoning over past events, while processing the full history at every control step can be costly. Two-stage architectures address this tension by using a VLM for memory-based subgoal selection and a VLA for low-level action generation (Sridhar et al., 2026; Physical Intelligence, 2026). MemER (Sridhar et al., 2026) uses visual memory to select relevant past observations and generate subgoal instructions. MEM (Physical Intelligence, 2026) updates textual long-term memory alongside high-level subgoal prediction. Our evaluation compares memory strategies within a shared two-stage framework, examining how their retained information supports different requests and subsequent execution (Section 4).

Embodied memory and task benchmarks. These advances motivate evaluation of which memory requirements different architectures support and how their memories contribute to task execution. Table 1 highlights differences in the memory demands, interaction capabilities, and evaluation protocols covered by existing benchmarks. MemoryBench (Fang et al., 2025), RoboMME (Dai et al., 2026), and RoboMemArena (Lei et al., 2026) test state and event memory through low-level manipulation, but do not combine manipulation with navigation. Combining the two creates memory demands across locations, as navigation takes previously observed objects out of view and manipulation changes their states. FindingDory (Yadav et al., 2026) and STARBench (Chen et al., 2025) evaluate memory-guided navigation and object retrieval, respectively, without evaluating low-level manipulation control. WorldLines (Zhang et al., 2026) evaluates memory QA and state-aware planning from structured household interaction histories, without evaluating visual memory construction or closed-loop task execution. While LongAct (Zhu et al., 2026) evaluates agents over histories generated during task execution, R&R-Bench provides validated experience contexts, enabling memory evaluation over long histories without requiring agents to successfully generate them themselves.

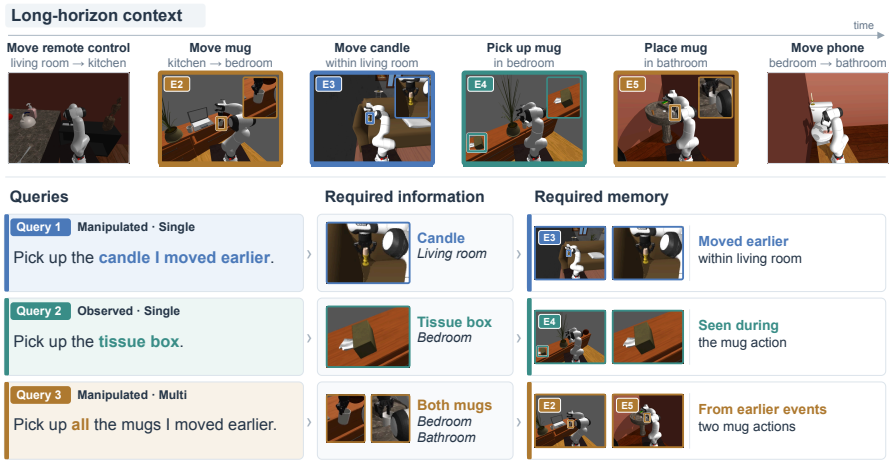

Figure 3: **Example Finding requests and their memory requirements.** Three Finding requests draw on the same recorded context: recalling a directly manipulated object (Query1), an incidentally observed object (Query2), or combining locations from two past events (Query3). Matching colors link context frames to the memories required by each request.

By combining long recorded experience with navigation and low-level manipulation, R&R-Bench evaluates memory construction and retrieval, planning and reasoning, including inferring rules from past experience, as well as online memory updating and task execution. Separate evaluation modes allow memory construction, retrieval, and reasoning to be assessed apart from embodied execution, while also evaluating online memory updating as agents act.

3 The R&R-Bench Benchmark

R&R-Bench combines four task families with diagnostic axes for experience source, request scope, and target visibility.

3.1 Contexts, Requests, and Evaluation References

Each benchmark instance includes a recorded context, a task request, a reference answer, an execution starting state, and task-specific success criteria. The context records the robot's observations and interactions, including object relocations and passively observed objects. Agents construct memory before receiving the request. Multiple requests can draw on the same context. Figure 3 illustrates how requests sharing the same context require different evidence, including directly manipulated objects, passively observed objects, and information spanning multiple events. Reference answers specify information needed for memory QA, such as target locations or unfinished work. Execution starts from the simulator state at the end of the provided context and is evaluated using the task-specific success criteria described in Section 3.2.

3.2 Task Families and Diagnostic Axes

Task families. We organize requests by how past experience informs action: locating a target, restoring an earlier state, continuing unfinished work, or applying a learned regularity. These families test complementary information requirements.

Finding requires identifying a requested object and recovering its last observed location. Targets may be concealed or unconcealed, and previously manipulated or merely observed.

Example

Context. The recorded robot placed a remote control inside a drawer that is closed at query time.

Request. "Pick up the remote control I moved earlier."

Evaluation criteria. The agent picks up the requested remote.

Restore requires recalling an object's current and earlier locations to return it to the earlier location.

Example

Context. The remote control was originally on a stand before being moved into the chest of drawers.

Request. "Pick up the remote control and place it in or on where it first was."

Evaluation criteria. The agent places the remote on its original stand.

Resume requires recovering interrupted work and distinguishing completed from remaining action units. Subsequent progress must be tracked so that completed work is not repeated.

Example

Context. The robot was instructed to move five potatoes from a dresser to a dining table. After moving one potato, it was asked to close a drawer and return. Four potatoes remain on the dresser.

Request. "You were interrupted earlier. Go back and finish what you were doing."

Evaluation criteria. The agent places the four remaining potatoes on the designated dining table.

Routine requires inferring and applying a storage rule from previous experience. It tests applying a regularity across experiences rather than recalling a previous location of the same object.

Example

Context. An egg, a potato, and an apple moved onto a side table, eating utensils moved onto a sofa.

Request. "Put the tomato on the countertop away."

Evaluation criteria. The agent places the tomato on the side table indicated by the recorded examples.

Request scope. Single and Multi indicate whether request execution involves one or multiple target objects or unfinished tasks. Multi requests require online memory updates to track completed and remaining work across targets or tasks, such as which cups have been restored. Single requests may also require online memory updates, such as remembering locations already searched.

Experience source distinguishes information about targets the recorded robot directly manipulated (Manipulated) from information obtained without direct interaction (Observed). Both are available as observations in the context video.

Target visibility distinguishes Concealed and Unconcealed targets. For example, an object inside a closed drawer is Concealed, whereas one on an open tabletop is Unconcealed.

3.3 Benchmark construction and validation

Context generation. We use MolmoSpaces (Kim et al., 2026) to generate and record experience in multi-room houses. We summarize each house into a scene card of rooms, objects, and receptacles.We then construct scenarios using predefined event structures for each task, randomly selecting objects and locations from eligible candidates in the scene card. These scenarios are executed using scripted navigation and manipulation routines. The scripted routines use A* (Hart et al., 1968) for navigation and precomputed grasp candidates from the MolmoSpaces grasp database, together with inverse kinematics and collision checking, for manipulation. The recordings capture the robot's observations and action outcomes, including unsuccessful actions, with associated task instructions.

Request generation. We generate requests using task-specific sentence templates filled with object and location names, and derive reference answers from the recorded states and action outcomes. Appendix A describes the context-generation pipeline and task-specific request construction.

Validation checks. Before recording, we check whether the robot can reach and manipulate the objects specified in the plan and whether placed objects remain stable in simulation. After recording, we exclude requests with missing evidence or unmet task-specific validity conditions. Query-time visibility checks and no-memory controls assess whether requests can be resolved from current observations, instruction wording, or common priors alone. Appendix A.3 provides validation details.

The resulting benchmark comprises 300 recorded contexts across 228 houses and 1,772 primary queries: 722 Finding, 575 Restore, 161 Resume, and 314 Routine. Figure 1 summarizes the task composition, context durations, and diversity of household environments.

4 EXPERIMENTS

Evaluation setting. Our evaluations connect two memory requirements, recalling experience from before a request (*retrospective memory*) and maintaining information acquired while carrying it out (*execution-state memory*). In **memory QA**, the VLM answers the request using stored memory and the current observation without acting. This evaluates how a system constructs memory from recorded context and retrieves and reasons over that memory to recover facts, prior states, events, and rules, without navigation or manipulation failures affecting the score. In **execution with ground-truth instructions**, the agent starts from a correct goal and acts while updating memory from new observations and execution progress. This removes errors in retrospective goal recall and evaluates progress tracking within the perception, planning, and control loop. Success rates and action-sequence metrics characterize behavior. **Execution with recalled instructions** evaluates the full pipeline, combining memory construction, goal recovery, and task execution with online updates.

Experimental setup. We evaluate a two-stage VLA system with memory. Before receiving a request, VLM writers sequentially process the recorded context to construct memory. The memory reader then receives the request, current observation, and the constructed memory. It returns a structured answer for QA or, during execution, selects subgoals and decides when to stop. During execution, the VLM writer observes action outcomes and updates memory with new observations and task progress. We use Gemini 3.7 Flash (Google DeepMind, 2026) as the memory writer and evaluate Gemini 3.7 Flash, Qwen3.8-27B (Qwen Team, 2026), and Gemma 4-31B (Gemma Team, 2026) as memory readers. We use fine-tuned MolmoBot (Deshpande et al., 2026) for manipulation (Appendix E). All reported results are averaged over three independent runs. Appendix B details the memory and execution interfaces.

Metrics. Memory QA accuracy is the fraction of evaluated queries answered correctly. Finding and Routine require the requested locations to match, while Restore requires both current and reference locations. These scores check rooms, surfaces, and answer counts. Resume requires the unfinished tasks and the number of remaining action units to match. Detailed scoring rules are provided in Appendix B.3. Execution success rate is the fraction of requests whose environmental goals are satisfied within the action budget. We report success over all queries and success among queries with correct memory QA answers separately.

Baselines. NO MEMORY uses no memory of past experience. GT ACTION LOG provides annotated records of the robot's actions and their outcomes. It is a reference for experience from manipulation and does not cover all facts acquired through observation. UNIFORM FRAMES samples historical frames uniformly. We reimplement MEM (Physical Intelligence, 2026), which repeatedly rewrites a text summary using prior memory and new observations. We also reimplement MemER (Sridhar et al., 2026), which selects frames to retain in visual memory. The symbol ^{R} denotes our reimplementations adapted to the shared evaluation framework. R&R AGENT is described in Section 5.2. Figure 4 compares the memory methods within the shared framework.

5 ANALYSIS

5.1 INSIGHTS FROM BASELINE COMPARISONS

We use R&R-Bench to examine how memory strategies perform across different information requirements and evaluation conditions. The following observations characterize memory demands through patterns among the evaluated methods and configurations, informing the reference design in Section 5.2.

1. Visual memory supports spatial recall, while textual memory supports summarization and reasoning over extended experience. In both memory QA and full-pipeline execution, visual memory (MEMER^{R}) is more effective for Finding and Restore, which require recalling specific object locations

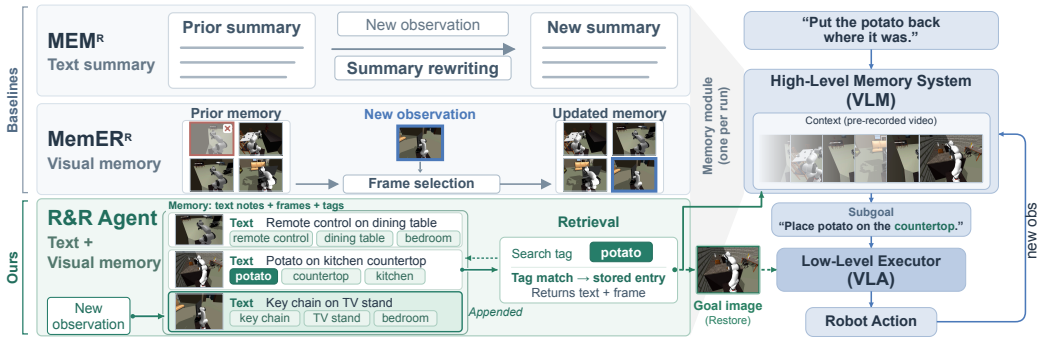

Figure 4: Memory methods within the shared two-stage VLA framework. A VLM reads memory to generate subgoals, and a VLA executes them. MEM^{R} and MEMER^{R} are our reimplementations that maintain text summaries and selected frames, respectively. R&R AGENT retains textual and visual evidence and retrieves relevant entries using tags. The main execution comparisons share the language-conditioned manipulation policy; the goal-image extension is evaluated separately in Table 6.

and earlier spatial states. Textual memory (MEM^{R}) performs better on Resume and Routine, which require consolidating past actions or inferring recurring rules from experience (Tables 2 and 4).

2. Visual memory helps retain passive observations. Information observed without direct interaction may become relevant to a later request. Visual memory generally performs better on Observed requests in Finding, Restore, and Routine, in both memory QA and full-pipeline execution (Tables 2 and 4).

3. Multi-objective requests challenge both recall and execution. Textual and visual memory methods show substantial performance drops from Single to Multi requests in both memory QA and full-pipeline execution (Tables 2 and 4). Multi requests require recovering more information and, during execution, coordinating progress across targets or tasks. These results motivate examining information recovery and execution behavior together when diagnosing multi-objective failures.

4. Memory representations perform differently across recall and execution conditions. In Finding and Restore, visual memory outperforms textual memory in memory QA, whereas textual memory achieves higher execution success when correct initial goals are provided (Tables 2 and 3). This suggests that preserving past evidence and maintaining information during execution impose different memory demands.

5. Textual instructions underspecify precise placement. Text alone can leave the intended placement underspecified, such as when returning an object to an earlier position in Restore (Table 6).

6. Request-conditioned retrieval is a promising design direction. The same context can support requests requiring different evidence. Summarizing observations or selecting frames before the request is known may discard details needed later. This motivates preserving a broader record of experience and retrieving relevant evidence after the request arrives, as explored in R&R AGENT.

5.2 R&R AGENT: A REFERENCE DESIGN

R&R AGENT combines textual records, visual evidence, and request-conditioned retrieval within the shared two-stage VLA framework (Figure 4). Implementation details are provided in Appendix B.

Complementary evidence for different requests. R&R AGENT maintains both textual records and visual evidence. Text makes actions, task progress, and recurring rules explicit; images retain scene details that can be inspected when the request reveals what matters. The System 2 VLM reader retrieves evidence to answer memory-QA requests or generate execution subgoals.

Preserve records and retrieve them when needed. R&R AGENT adds retained records and images to a database without replacing earlier entries with a single updated summary. Each entry carries time information and tags for objects, actions, and locations. For each request, the reader uses these tags to retrieve relevant text records and images. This preserves access to earlier stored evidence,

Table 2: Memory QA accuracy (%) by task and subtype for three VLMs. ^{R} denotes our reimplementations adapted to the shared evaluation framework.

Strategy	Finding						Restore						Resume			Routine				
	S-Man.	S-Obs.	S-Conc.	M-Man.	M-Obs.	**Avg.**	S-Man.	S-Obs.	S-Conc.	M-Man.	M-Obs.	**Avg.**	Single	Multi	**Avg.**	S-Man.	S-Obs.	M-Man.	M-Obs.	**Avg.**
Gemini 3.7 Flash																				
GT Action Log	85.5	14.0	71.4	63.9	2.0	46.4	90.2	0.0	89.3	59.5	0.0	46.3	100.0	100.0	100.0	100.0	2.6	100.0	0.0	75.8
No Memory	19.5	17.0	0.0	0.0	4.0	12.0	0.9	0.0	0.0	0.0	0.0	0.3	10.0	1.6	6.8	11.5	5.3	0.0	2.6	7.4
Uniform Frames	38.2	55.9	26.5	8.4	21.0	35.7	16.7	8.2	7.1	1.3	0.0	9.6	33.0	14.8	26.1	29.9	28.9	6.7	5.3	22.3
MEM^{R}	42.7	42.8	**57.1**	16.0	5.0	34.0	23.7	3.6	21.4	2.5	0.0	11.5	53.0	34.4	46.0	74.1	18.4	50.0	5.3	54.2
MemER^{R}	50.5	**63.3**	40.8	10.9	27.0	44.1	26.0	13.3	14.3	**6.3**	**1.8**	16.0	39.0	16.4	30.4	39.7	**34.2**	8.3	5.3	28.7
R&R Agent	**54.5**	59.4	**57.1**	**16.8**	**30.0**	**46.6**	**28.4**	**17.3**	**32.1**	5.1	**1.8**	**19.0**	**55.0**	**49.2**	**52.8**	**78.2**	**34.2**	**60.0**	**7.9**	**60.6**
Qwen3.8-27B																				
GT Action Log	85.3	11.8	79.2	64.7	2.0	46.1	85.4	0.5	85.7	59.0	0.0	44.3	92.0	96.7	93.8	100.0	0.0	98.3	0.0	75.2
No Memory	17.5	15.3	2.1	0.8	7.0	11.5	2.4	1.5	0.0	0.0	0.0	1.4	19.0	6.6	14.3	0.6	0.0	0.0	0.0	0.3
Uniform Frames	23.0	33.6	4.2	1.7	9.0	19.6	7.5	3.6	0.0	1.3	0.0	4.2	22.0	11.5	18.0	6.3	0.0	0.0	0.0	3.5
MEM^{R}	18.0	19.2	10.4	0.8	2.0	12.8	5.7	2.6	7.1	0.0	0.0	3.3	**54.0**	26.2	43.5	54.0	0.0	26.7	0.0	35.5
MemER^{R}	24.5	38.4	14.3	5.7	8.0	22.8	8.0	5.1	0.0	0.0	0.0	4.7	25.0	9.8	19.3	8.0	**13.2**	0.0	0.0	6.1
R&R Agent	**48.6**	**53.3**	**51.0**	**9.8**	**16.0**	**39.2**	**26.4**	**11.8**	**25.0**	**5.1**	**1.8**	**16.0**	**54.0**	**31.1**	**45.3**	**64.4**	7.9	**35.0**	**2.6**	**44.2**
Gemma 4-31B																				
GT Action Log	84.8	8.7	72.9	64.7	1.0	44.5	84.0	0.0	82.1	60.3	0.0	43.6	92.0	91.8	91.9	100.0	0.0	95.0	0.0	74.5
No Memory	18.0	27.9	10.4	0.8	5.0	16.0	4.2	1.5	3.6	0.0	0.0	2.3	5.0	3.3	4.3	1.1	0.0	0.0	0.0	0.6
Uniform Frames	24.0	46.3	16.7	2.5	14.0	25.7	6.6	2.1	0.0	0.0	0.0	3.2	14.0	1.6	9.3	6.9	0.0	0.0	0.0	3.9
MEM^{R}	16.6	23.6	18.8	3.4	4.0	15.0	7.5	2.1	7.1	0.0	0.0	3.9	34.0	11.5	25.5	62.1	2.6	30.0	0.0	41.0
MemER^{R}	35.9	48.5	28.6	3.3	11.0	30.4	11.8	4.1	7.1	1.3	0.0	6.3	20.0	3.3	13.7	5.7	2.6	0.0	0.0	3.5
R&R Agent	**47.7**	**54.6**	**51.0**	**10.7**	**15.0**	**39.3**	**27.4**	**15.9**	**21.4**	**2.6**	**1.8**	**17.2**	**40.0**	**23.0**	**33.5**	**70.1**	**15.8**	**50.0**	**2.6**	**51.3**

including reference states needed for Restore. During execution, the writer adds observations and progress information so that later decisions can distinguish completed from remaining work.

Visual evidence for precise action. Retrieved goal images can convey spatial details that textual subgoals leave unspecified. We explore this extension separately from the main execution comparisons, which use a shared language-conditioned manipulation policy.

5.3 Evaluation evidence and further analysis

Memory QA. Table 2 evaluates memory construction, retrieval, and reasoning without VLA execution. Across all three readers, MemER^{R} leads in Finding and Restore, while MEM^{R} leads in Resume and Routine. For Gemini 3.7 Flash, MemER^{R} achieves 44.1% vs 34.0% in Finding, whereas MEM^{R} achieves 54.2% vs 28.7% in Routine. Within Routine, Single Manipulated queries favor MEM^{R}, while Single Observed queries favor MemER^{R}.

The benefit of retaining images is visible without learned frame selection: Uniform Frames exceeds MEM^{R} on Finding with every reader. This is consistent with later requests requiring details that a prior summary may omit. Combining records and images, R&R Agent achieves the highest task-average QA accuracy among methods that construct memory from experience for every task and reader.

Execution with ground-truth instructions. Table 3 evaluates execution-state memory together with task execution, given correct initial goals and no memory from the recorded context. Success rate alone does not distinguish successful execution with accurate progress tracking from success achieved with redundant actions. For example, a task requiring subtasks `ABC` may still be scored as successful after `AADBBECCC`, despite repeating completed subtasks and performing unrelated actions `D` and `E`, as long as the final goal conditions are satisfied. We therefore complement success rate with action-sequence metrics that help assess how agents track progress during execution.

Edit distance (ED) measures the Levenshtein distance between the executed and ideal action sequences, normalized by the ideal sequence length and averaged over queries. It captures insertions, deletions, and substitutions. Ins. measures the average number of extra actions per request, including actions on already-completed work and unnecessary visits. Lower values indicate closer adherence to the required task sequence. The scoring consolidates designated retry patterns so that repeated low-level attempts are not all counted as separate actions.

Textual memory (MEM^{R}) achieves lower ED and fewer extra actions than visual memory (MemER^{R}) across all four task families. This supports explicit textual progress summaries when retrospective goal recovery is unnecessary. R&R Agent further reduces extra actions relative to MEM^{R} in Restore, Resume, and Routine, while matching it in Finding.

Table 3: Execution with ground truth instructions (all runs), using Gemini 3.7 Flash and MolmoBot. SR: success rate (%); ED and Ins.: see Section 5.3. Scoring details are in Appendix B.3. Bold and underlining mark the best and next distinct values.

Strategy	Finding			Restore			Resume			Routine		
	Ins.↓	ED↓	SR↑	Ins.↓	ED↓	SR↑	Ins.↓	ED↓	SR↑	Ins.↓	ED↓	SR↑
NO MEMORY	0.85	0.660	36.7	2.01	0.766	10.8	3.51	0.919	18.6	1.50	0.755	15.4
UNIFORM FRAMES	0.40	0.568	**38.7**	1.84	0.767	14.2	2.08	0.667	26.1	1.34	0.792	12.2
MEM^R	**0.30**	**0.555**	35.3	1.44	0.687	**19.6**	1.43	**0.484**	**30.4**	1.09	0.728	16.3
MEMER^R	0.70	0.636	32.7	1.64	0.713	16.2	2.15	0.723	21.1	1.40	0.781	15.4
R&R AGENT	**0.30**	0.575	36.0	**1.12**	**0.632**	17.6	**1.14**	0.513	29.8	**0.85**	**0.689**	**17.9**

Execution with recalled instructions. Table 4 reports full-pipeline performance, from memory construction and goal recovery by the VLM reader to task execution. R&R AGENT has the highest task-average success across all four task families among methods without ground-truth action records. Baseline differences also persist: MEMER^R exceeds MEM^R on Single Observed Finding queries (39.7% versus 27.9%), while MEM^R leads on average in Resume and Routine.

Table 4: Execution with recalled instructions (success rate, %).

Strategy	Finding						Restore						Resume			Routine				
	S-Man.	S-Obs.	S-Conc.	M-Man.	M-Obs.	**Avg.**	S-Man.	S-Obs.	S-Conc.	M-Man.	M-Obs.	**Avg.**	Single	Multi	**Avg.**	S-Man.	S-Obs.	M-Man.	M-/Obs.	**Avg.**
GT ACTION LOG	51.8	10.0	14.0	4.1	0.0	20.6	21.4	0.0	0.0	1.3	0.0	8.2	32.0	1.6	20.5	25.3	0.0	0.0	0.0	17.4
NO MEMORY	8.6	11.4	0.0	0.0	0.0	6.2	0.5	0.0	0.0	0.0	0.0	0.2	0.0	0.0	0.0	2.9	0.0	0.0	0.0	2.0
UNIFORM FRAMES	23.6	37.6	4.0	**1.6**	3.0	20.1	3.7	2.0	0.0	0.0	0.0	2.1	4.0	0.0	2.5	6.9	**13.2**	0.0	0.0	6.7
MEM^R	25.9	27.9	**10.0**	0.0	0.0	17.5	7.4	0.0	0.0	0.0	0.0	2.8	11.0	**4.9**	8.7	14.4	2.6	0.0	0.0	10.3
MEMER^R	28.6	**39.7**	6.0	0.8	8.0	23.0	6.0	4.1	0.0	0.0	0.0	3.7	9.0	0.0	5.6	9.8	7.9	0.0	0.0	7.9
R&R AGENT	**35.0**	36.7	**10.0**	**1.6**	**9.0**	**24.5**	**11.2**	**4.6**	0.0	0.0	0.0	**5.7**	**15.0**	**4.9**	**11.2**	**17.2**	**13.2**	0.0	0.0	**13.8**

The source of Multi difficulty differs by task. Multi Manipulated Restore QA is at most 6.3% among constructed-memory methods. In Routine, MEM^R and R&R AGENT reach 50.0% and 60.0% QA accuracy on Multi Manipulated queries, yet neither completes these requests. Thus, low execution success can coexist with either poor information recovery or substantial QA accuracy.

Complementary representations. Table 5 compares R&R AGENT's memory variants. Text-only memory matches or exceeds MEM^R across all tasks. Combining text and frames improves Restore, Resume, and Routine over either modality alone, with the largest gain in Resume (52.8% vs 46.0% for text only),supporting the complementary value of textual and visual evidence in these tasks.

Table 5: Memory modality ablation (Gemini 3.7 Flash, accuracy, %). Same QA scoring and eligible queries. Bold: column best.

Method	Finding	Restore	Resume	Routine
Text-based memory				
MEM^R	34.0	11.5	46.0	54.2
R&R AGENT (text only)	42.0	16.2	46.0	58.7
Frame-based memory				
UNIFORM FRAMES	35.7	9.6	26.1	22.3
MEMER^R	44.1	16.0	30.4	28.7
R&R AGENT (frames only)	**47.7**	15.7	22.4	30.6
Text + frames				
R&R AGENT (full)	46.6	**19.0**	**52.8**	**60.6**

Visual goals for precise manipulation. Table 6 evaluates precise placement in Restore, returning objects to their earlier positions. Trials start with the object grasped and the robot at the correct destination. The max-area frame shows the target object largest in the recorded history. The target render depicts the current scene with the object at its desired location from a random camera direction. All goal-image conditions have numerically higher placement scores than no image, with R&R AGENT selected images performing best. (Appendix C.5).

Table 6: Goal conditioning (%). On: correct support. @5/10/15: Replace accuracy. Bold: column best.

Goal image source	On↑	@5↑	@10↑	@15↑
No goal image	52.7	1.1	5.4	9.7
Max-area frame	58.1	4.3	9.7	19.4
Target render	58.1	3.2	9.7	17.2
R&R AGENT (ours)	**59.1**	**5.4**	**11.8**	**20.4**

6 CONCLUSION AND LIMITATIONS

We introduced R&R-Bench to evaluate diverse memory requirements and their relationship to embodied task execution. Its task variations and complementary evaluation conditions characterize differences across memory strategies and reveal execution challenges that remain even when correct goals are available. Guided by these findings, R&R AGENT combines textual records, visual evidence, and request-conditioned retrieval as a reference design. A separate goal-image experiment examines how retrieved visual evidence can support precise placement. Our evaluation is limited to simulated households and the tested models and configurations; generalization to physical robots and broader environments remains open.

AI USE STATEMENT

We used generative AI tools (Claude Code and Codex) to assist with software implementation, evaluation and analysis scripts, and manuscript editing. The authors determined the research questions, benchmark and experimental design, provided direction and feedback, interpreted the results, and reviewed all scientific claims. Generative AI was not used to generate experimental outcomes. All reported quantitative values were checked against the underlying experiment logs, and the authors take responsibility for the final text, claims, results, and artifacts.

ETHICS STATEMENT

R&R-Bench is built entirely in simulation from procedurally generated houses and publicly available 3D assets. It involves no human subjects, no personally identifiable information, and no real-world data collection. We propose the benchmark for evaluation purposes. Although the benchmark itself contains no personal data, household-robot memory systems deployed in real homes raise privacy and data-governance questions that simulated evaluation does not capture.

REPRODUCIBILITY STATEMENT

All context episodes, queries, ground truth, and the evaluation harness (ingestion, memory strategies, scoring, and the online execution loop) will be released. Section 3.3 describes the generation pipeline, Section 4 the exact scoring rules, and the appendix lists prompts and per-family scoring contracts.

REFERENCES

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: A vision-language-action model with open-world generalization. In *Conference on Robot Learning (CoRL)*, 2025.

Taijing Chen, Sateesh Kumar, Junhong Xu, Georgios Pavlakos, Joydeep Biswas, and Roberto Martín-Martín. Searching in space and time: Unified memory-action loops for open-world object retrieval. *arXiv preprint arXiv 2511.14004*, 2025.

Yinpei Dai, Hongze Fu, Jayjun Lee, Yuejiang Liu, Haoran Zhang, Jianing Yang, Chelsea Finn, Nima Fazeli, and Joyce Chai. RoboMME: Benchmarking and understanding memory for robotic generalist policies. In *International Conference on Machine Learning (ICML)*, 2026.

Samyak Datta, Sameer Dharur, Vincent Cartillier, Ruta Desai, Mukul Khanna, Dhruv Batra, and Devi Parikh. Episodic Memory Question Answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19119–19128, June 2022. URL https://openaccess.thecvf.com/content/CVPR2022/html/Datta_Episodic_Memory_Question_Answering_CVPR_2022_paper.html.

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. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Abhay Deshpande, Maya Guru, Rose Hendrix, Snehal Jauhri, Ainaz Eftekhar, Rohun Tripathi, Max Argus, Jordi Salvador, Haoquan Fang, Matthew Wallingford, Wilbert Pumacay, Yejin Kim, Quinn Pfeifer, Ying-Chun Lee, Piper Wolters, Omar Rayyan, Mingtong Zhang, Jiafei Duan, Karen Farley, Winson Han, Eli Vanderbilt, Dieter Fox, Ali Farhadi, Georgia Chalvatzaki, Dhruv Shah, and Ranjay Krishna. MolmoB0T: Large-Scale Simulation Enables Zero-Shot Manipulation. *arXiv preprint arXiv:2603.16861*, 2026. URL https://arxiv.org/abs/2603.16861.

Haoquan Fang, Markus Grotz, Wilbert Pumacay, Yi Ru Wang, Dieter Fox, Ranjay Krishna, and Jiafei Duan. SAM2Act: Integrating visual foundation model with a memory architecture for robotic manipulation. In *International Conference on Machine Learning (ICML)*, 2025.

Gemma Team. Gemma 4 Technical Report. *arXiv preprint arXiv:2607.02770*, 2026. URL https://arxiv.org/abs/2607.02770.

Google DeepMind. Gemini 3.7 Flash, 2026. URL https://deepmind.google/models/model-cards/gemini-3-7-flash/. Model card.

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.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner Monologue: Embodied Reasoning through Planning with Language Models. In Karen Liu, Dana Kulic, and Jeff Ichnowski (eds.), *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782. PMLR, 2023. URL https://proceedings.mlr.press/v205/huang23c.html.

Yejin Kim, Wilbert Pumacay, Omar Rayyan, Max Argus, Winson Han, Eli VanderBilt, Jordi Salvador, Abhay Deshpande, Rose Hendrix, Snehal Jauhri, Shuo Liu, Nur Muhammad Mahi Shafiullah, Maya Guru, Ainaz Eftekhar, Karen Farley, Donovan Clay, Jiafei Duan, Arjun Guru, Piper Wolters, Alvaro Herrasti, Ying-Chun Lee, Georgia Chalvatzaki, Yuchen Cui, Ali Farhadi, Dieter Fox, and Ranjay Krishna. MolmoSpaces: A large-scale open ecosystem for robot navigation and manipulation. *arXiv preprint arXiv 2602.11337*, 2026.

Myungkyu Koo, Daewon Choi, Taeyoung Kim, Kyungmin Lee, Changyeon Kim, Younggyo Seo, and Jinwoo Shin. HAMLET: Switch your vision-language-action model into a history-aware policy. In *International Conference on Learning Representations (ICLR)*, 2026.

Huashuo Lei, Wenxuan Song, Huarui Zhang, Jieyuan Pei, Jiayi Chen, Haodong Yan, Han Zhao, Pengxiang Ding, Zhipeng Zhang, Lida Huang, Donglin Wang, Yan Wang, and Haoang Li. RoboMemArena: A comprehensive and challenging robotic memory benchmark. *arXiv preprint arXiv:2605.10921*, 2026.

Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Fei Han, Mingrui Yu, Zelin Gao, Nan Xue, Xing Zhu, Yujun Shen, and Yinghao Xu. Causal world modeling for robot control. In *Robotics: Science and Systems (RSS)*, 2026.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. *arXiv preprint arXiv:1711.05101*, 2017. URL https://arxiv.org/abs/1711.05101.

Physical Intelligence. Memory for generalist robot policies. https://www.pi.website/download/Mem.pdf, 2026. Technical report.

Physical Intelligence, Bo Ai, Ali Amin, et al. $\pi_{0.7}$: a Steerable Generalist Robotic Foundation Model with Emergent Capabilities. *arXiv preprint arXiv:2604.15483*, 2026. URL https://arxiv.org/abs/2604.15483.

Qwen Team. Qwen3.8-27B, 2026. URL https://huggingface.co/Qwen/Qwen3.8-27B. Model card.

Hao Shi, Bin Xie, Yingfei Liu, Lin Sun, Fengrong Liu, Tiancai Wang, Erjin Zhou, Haoqiang Fan, Xiangyu Zhang, and Gao Huang. MemoryVLA: Perceptual-cognitive memory in vision-language-action models for robotic manipulation. In *International Conference on Learning Representations (ICLR)*, 2026.

G. Smith, S. Della Sala, R. H. Logie, and E. A. Maylor. Prospective and retrospective memory in normal ageing and dementia: A questionnaire study. *Memory*, 8(5):311–321, 2000. doi: 10.1080/09658210050117735.

Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. MemER: Scaling up memory for robot control via experience retrieval. In *International Conference on Learning Representations (ICLR)*, 2026.

Haiyang Sun, Haoxiao Wang, Junming Chen, Weicheng Fang, Zihao Su, Jingkun Yi, Wenyou Yi, Hao Chen, and Zhou Zhao. MEMOBench: A process level memory benchmark for robotic manipulation. *arXiv preprint arXiv 2609.07047*, 2026.

Liufan Tan, Jiale Li, and Gangshan Jing. MemoAct: Atkinson-shiffrin-inspired hierarchical memory-augmented policy for robotic manipulation. *IEEE Robotics and Automation Letters*, 11(9):10927–10934, 2026.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

Luca Weihs, Matt Deitke, Aniruddha Kembhavi, and Roozbeh Mottaghi. Visual Room Rearrangement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5922–5931, June 2021. URL https://openaccess.thecvf.com/content/CVPR2021/html/Weihs_Visual_Room_Rearrangement_CVPR_2021_paper.html.

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. *Autonomous Robots*, 47(8):1087–1102, 2023. doi: 10.1007/s10514-023-10139-z. URL https://link.springer.com/article/10.1007/s10514-023-10139-z.

Karmesh Yadav, Yusuf Ali, Gunshi Gupta, Yarin Gal, and Zsolt Kira. FindingDory: A benchmark to evaluate memory in embodied agents. In *International Conference on Machine Learning (ICML)*, 2026.

Yehang Zhang, Jianchong Su, Haojian Huang, Yifan Chang, Tianhao Zhou, Xinli Xu, Yingjie Xu, Yinchuan Li, Zexi Li, and Ying-Cong Chen. WorldLines: Benchmarking and modeling long-horizon stateful embodied agents. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2026.

Zilin Zhu, Longteng Guo, Yanghong Mei, Bowen Pang, Zongxun Zhang, Xingjian He, Ruyi Ji, and Jing Liu. When robots do the chores: A benchmark and agent for long-horizon household task execution. *arXiv preprint arXiv 2605.14504*, 2026.

APPENDIX CONTENTS

A Benchmark details **14**

A.1 Context generation . . . 14

A.2 Task-specific requests . . . 15

A.3 Validation and memory-dependence checks . . . 17

A.4 Dataset composition . . . 18

B Evaluation details and metrics **18**

B.1 From experience to answer . . . 18

B.2 Acting and updating memory . . . 18

B.3 Evaluation metrics and scoring . . . 20

B.4 How R&R AGENT maintains memory . . . 20

C Additional experimental results **21**

C.1 Memory representations across information requirements . . . 21

C.2 Recovering both locations in Restore . . . 21

C.3 Visual traces of memory use . . . 21

C.4 Examples of stored memory . . . 22

C.5 VLM selection of goal images from memory . . . 22

C.6 Memory image budget . . . 22

C.7 Recovery after incorrect recall . . . 22

D Prompts Format **26**

E VLA fine-tuning and results **27**

E.1 Training configuration . . . 27

E.2 Manipulation abilities after fine-tuning . . . 27

E.3 Goal-image conditioning . . . 27

A BENCHMARK DETAILS

This section expands the context and request construction in Section 3.3, including task-specific selection rules.

A.1 CONTEXT GENERATION

Scene cards and house selection. The generation pipeline uses scene cards to support scenario generation and applies task-specific sampling and feasibility checks before simulation execution. First, each ProcTHOR (Deitke et al., 2022) house is analyzed once in MuJoCo (Todorov et al., 2012) into a *scene card*: rooms, room-to-room doors with their widths and pairwise geodesic distances, support surfaces, free-joint objects, and fixture compartments. Candidate houses are the 3,494 train houses of ProcTHOR-10k from MolmoSpaces, preferring three- to four-room layouts, of which 1,600 are converted to cards. A house passes the card gate only if it has three to four rooms of distinct types connected by doors, every anchor pair is navigable, every room has at least one reachable surface and one manipulable object, and the total geodesic length is at least 12 m.

Feasibility screening. We test grasping and insertion feasibility in the initial scene before constructing an event plan. For each candidate object, including distractors, the grasp planner searches for a complete approach and lift using 16 base poses and 30 grasp candidates. For each compartment, the insertion planner tests whether an object can be placed through its opening. Only candidates that pass the relevant checks are sampled. Because object additions and clutter can change feasibility, these checks are repeated during staging and execution. Episodes that fail task-specific validity criteria are discarded.

Scenario sampling. Scenarios are constructed using manually specified rules that define the event structure and constraints for each task. Given a scene card, the sampler uses seeded randomness to apply those rules without running the simulator. It determines the information needed for each task's scenario plan, including event or unit order, an estimated length within the duration window, instruction-template slots, and a starting room. The task-specific choices are:

- For *Finding*, the sampler fixes the object body for each event, its current room and support surface, and the destination room and surface (or fixture and compartment site for storage). The plan also fixes event tags, near-placement relations, and distractors.
- *Restore* reuses Finding's contexts and derives requests for earlier locations from recorded object displacements.
- For *Resume*, the sampler fixes each task's room, injected object asset type and counts, candidate source surfaces, destination type (basket or surface), and near/far relation. Staging selects the actual surfaces. Errand targets are fixed to specific fixtures or objects.
- For *Routine*, the sampler fixes a source table instance, a destination class for each storage rule, injected actor and holdout assets, and an ordered list of candidate surfaces for observation evidence. The plan does not name a destination instance. Staging resolves the evidence surfaces.

Concrete coordinates, manipulation poses, and navigation paths are determined during execution.

Scenario-guided action rollouts. For each scene, we execute shared navigation, grasping, carrying, and placement primitives according to its sampled scenario plan and record the resulting action trajectory. When each house's scene card is built, a preliminary pick-feasibility estimate (`pick_ok`) is recorded for candidate objects before scenario sampling. Navigation uses A* on a 5 cm occupancy grid with obstacle inflation and collision checks. Grasping uses precomputed grasp candidates and inverse kinematics. Placement selects a clear spot on the target surface. Before each primitive runs, its planner generates the scene-specific poses, joint-space trajectory, navigation path, or placement point it needs and checks candidate feasibility. For pick, it pairs up to 180 docking poses with database grasps and selects the first pair with a reachable, collision-free base pose and a feasible arm approach. Carry, place, and fixture primitives use the same generate-and-test approach. The robot then interpolates the planned key poses or waypoints at 25 Hz with cosine easing, without resolving inverse kinematics or applying feedback corrections during the primitive. Success is judged from the resulting simulator state rather than the commanded motion: pick succeeds only if the

object is lifted and remains near the gripper, and place only if it settles stably on the target surface. These outcome conditions and the primitive procedures are shared across scenes.

Failures as data. In Finding/Restore contexts, when a grasp slips or an object tumbles to the floor, the episode is retained and the affected answers are relabeled (incorrect placement, per-answer scoring downgraded to navigation-only), mirroring how real household histories contain failed actions. Routine contexts likewise retain failed units when the household rule remains supported: 355 of 426 units succeed in the release.

Robot and observation streams. The robot is a mobile Franka (7-DoF arm on a holonomic base) controlled at 25 Hz with base $[x, y, \mathrm{yaw}]$, seven arm joints, and a binary gripper. Three cameras (two exocentric, one wrist) record at the same 25 fps, so the k-th control step in the trajectory log corresponds to the k-th frame of every video, and frame indices serve as the common time axis for actions, observations, and context cuts.

A.2 TASK-SPECIFIC REQUESTS

Instruction and request templates. Task instructions and request sentences are produced from a few string templates per family (e.g., "Pick up {n} of the {obj}s from the {src} and place them in or on the {dst}."), whose slot words are derived from the MolmoSpaces category names in the scene card ("SideTable" → "side table"). Slots known at planning time (counts, object class, errand target) are filled during scenario sampling, while surface words are filled during execution after the actual surface is selected. Sentences deliberately name only classes, never instances, compartment ordinals, the number of answers in a multi request, or, for Resume and Routine, the task name or the storage rule. These omissions limit shortcuts based on request wording alone, leaving the unspecified instance, answer count, task, or rule to be resolved from the recorded context.

Task-family requests. Figure 5 shows examples from Restore, Resume, and Routine.

- **Finding and Restore.** Finding and Restore share contexts containing repeated object relocations and passively observed objects. Their requests refer to later recorded locations and earlier reference states, respectively. The contexts include same-class objects and closed compartments to limit shortcuts from the current scene.

- **Resume.** Resume contexts interleave household activities with unrelated errands and stop before the activities are complete. Requests ask the agent to resume the unfinished work from the end of the provided context. Activities include room visits and button presses without persistent visible traces, and relocation or storage into destinations already containing same-class objects. These constraints limit inference of task progress from the current scene alone.

- **Routine.** Routine has its own 100 contexts, each with two or three household storage rules and three or four queries. Storage rules are randomized to limit shortcuts from common priors, and query targets are withheld from demonstrations. Commands ask the robot to put objects away without naming the rule's destination. Rule values must be inferred from placements. Forty contexts also contain an observation-only rule, demonstrated by three objects already on their receptacle. Queries test held-out object subcategories using either a single-object request or a request to clear the start table. For example, a rule demonstrated with some members of an object group must be applied to another member of that group. Each context ends with a return-to-table segment, so the query begins facing the holdouts.

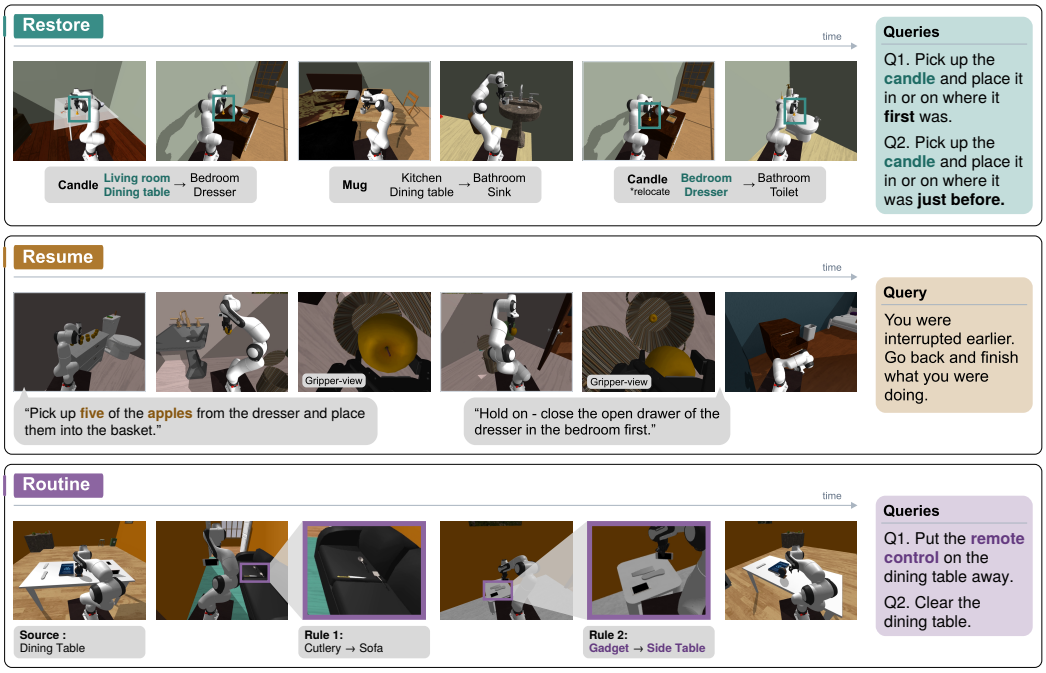

Figure 5: Restore, Resume, and Routine examples. Frames progress from left to right. Evaluation requests appear at right.

Figure 6 illustrates the diagnostic axes defined in Section 3.2 for Finding requests. The examples show Single/Multi request scope, Manipulated/Observed experience source, and a Concealed target hidden at query time.

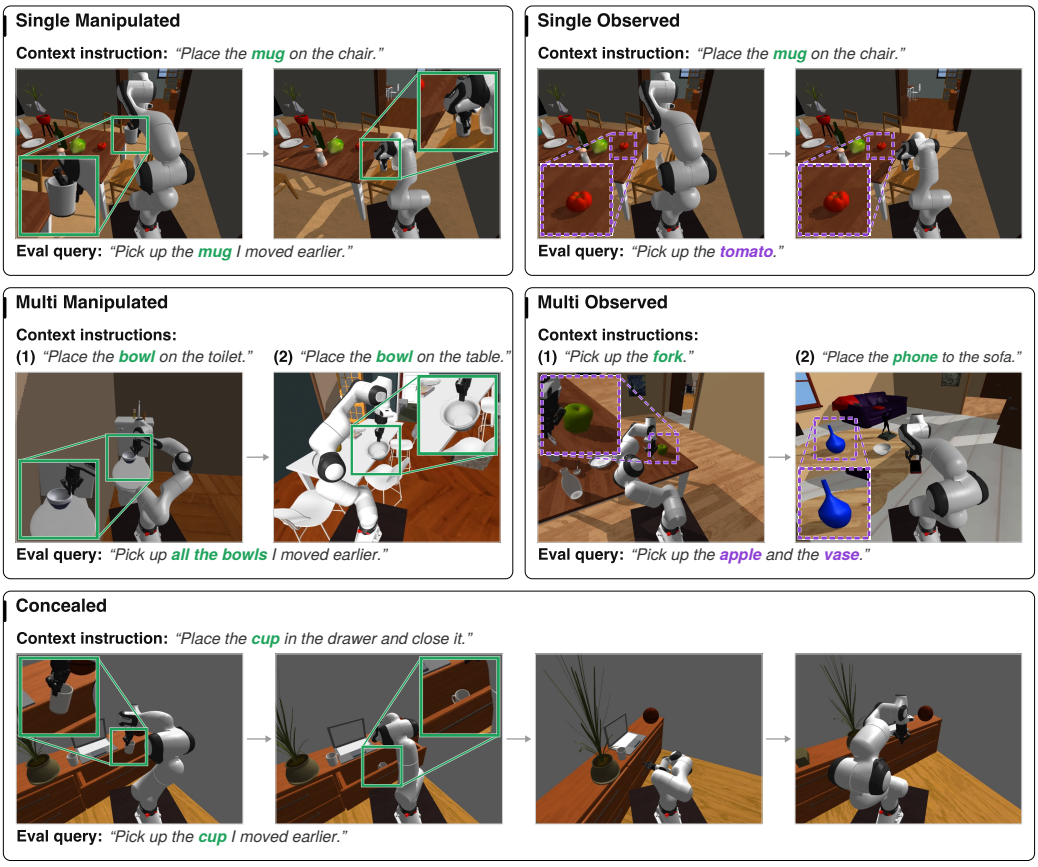

Figure 6: Finding examples of request scope, experience source, and target visibility.

A.3 Validation and memory-dependence checks

Reference annotations. Reference answers are read from the recorded state at the relevant time, not from the plan, so a failed scripted event is reflected in the answer. Recordings and requests that do not meet the following conditions are dropped automatically.

Episode validity. An episode is discarded if its record contract fails (frame counts, resolution, tracked trajectories, joint logs, phase labels, event timings), if a non-target object was dragged more than 25 cm during a unit, if simulation ran below 1.5 fps, if it is shorter than 300 s, or, for Finding, if more than one event failed or fewer than five requests survive.

Request validity. A request is discarded if its answer was never captured on camera during the context ($\geq$ 50 px), if its wording is ambiguous against a same-name distractor, if the target was an object the robot failed to pick, if a concealed answer is not behind a closed door at the end, or if the answer cannot be executed from the starting state, checked by running the pick and place planners at the answer location (for Restore: currently graspable, displaced $\geq$ 1 m, origin not since occupied, origin observed $\geq$ 1 s, placeable at the origin).

Resume and Routine checks. Resume cuts additionally require that the snapshot restores within 1 mm, that an oracle continuation completes the pending work, and that no post-cut instruction is included. Routine requests require the rule to be established in the recording (a successful unit, or

all three observed evidence objects visible at ≥ 200 px), the holdout still on the source table, and no counterexample in the final state.

Temporal leakage and query-time visibility. A leak check confirms that no loader output references frames beyond the context cut, and each request is labeled with whether its answer is visible from the starting state, which is reported but not used for exclusion.

A.4 Dataset composition

The release contains 300 contexts across 228 houses and 1,772 primary queries: 722 Finding, 575 Restore, 161 Resume, and 314 Routine queries. Finding and Restore share 100 contexts. Resume and Routine each use a separate set of 100. An additional 322 Resume cue variants are reserved for ablations and excluded from the primary total. The memory-QA results use the eligible queries defined for each task.

Finding contains 388 Manipulated and 334 Observed queries. Restore contains 322 and 253. Routine contains 234 Manipulated and 80 Observed queries. Executed rules require at least one successful placement, observed rules require visible evidence, and contexts with a final arrangement contradicting a rule are excluded. Concealment is a separate axis rather than a third evidence source: Single (Concealed) reporting cells pool evidence sources, while concealed multi-object requests remain in the corresponding Multi cells.

B Evaluation details and metrics

Harness overview. The evaluation follows a simple sequence: observe past experience, answer a request from memory, and, in the execution conditions, act while maintaining information about progress. Figure 7 summarizes this flow. Appendix D explains the prompt structure and memory-update instructions. Appendix B.4 summarizes how R&R Agent uses the same interface.

B.1 From experience to answer

Before the request, writers process the recorded experience in 30-second windows. The common writer instruction asks them to preserve what the robot did and saw, including objects encountered without being manipulated. Each method then applies its own storage rule: rewrite a summary, select frames, or add tagged memories. The writer is not given the future query. No Memory stores no history. GT Action Log provides privileged annotated action records rather than a learned memory and does not cover all passively observed facts.

At query time, the reader receives the request, the current observation, and the memory available under its method. It returns a structured answer without taking actions. Finding asks for target locations, Restore for current and previous locations, Resume for unfinished tasks and remaining counts, and Routine for placement rules.

B.2 Acting and updating memory

Execution restores the simulator to the request-time state and starts from either a supplied ground truth instruction or an instruction recovered by memory QA. The agent then chooses the actions needed to achieve that goal.

Figure 8 shows the interface supplied to the controller: available skills, house-specific object and room names, and the required JSON response. The agent copies names from these lists and returns one next subgoal together with its method's memory update.

At each decision point, the VLM receives the goal, recent frames, available memory, and object/room names. It returns one next subgoal together with the memory-update fields used by its method. The harness applies the update and executes the subgoal: navigation uses an occupancy-map planner, while manipulation uses the fine-tuned MolmoBot policy (Appendix E).

New observations return to the same loop. The controller is instructed to judge whether an action worked from the visual evidence, retry when needed, and declare completion only when the goal is satisfied. The run ends when the agent declares completion, aborts, or reaches its execution limit.

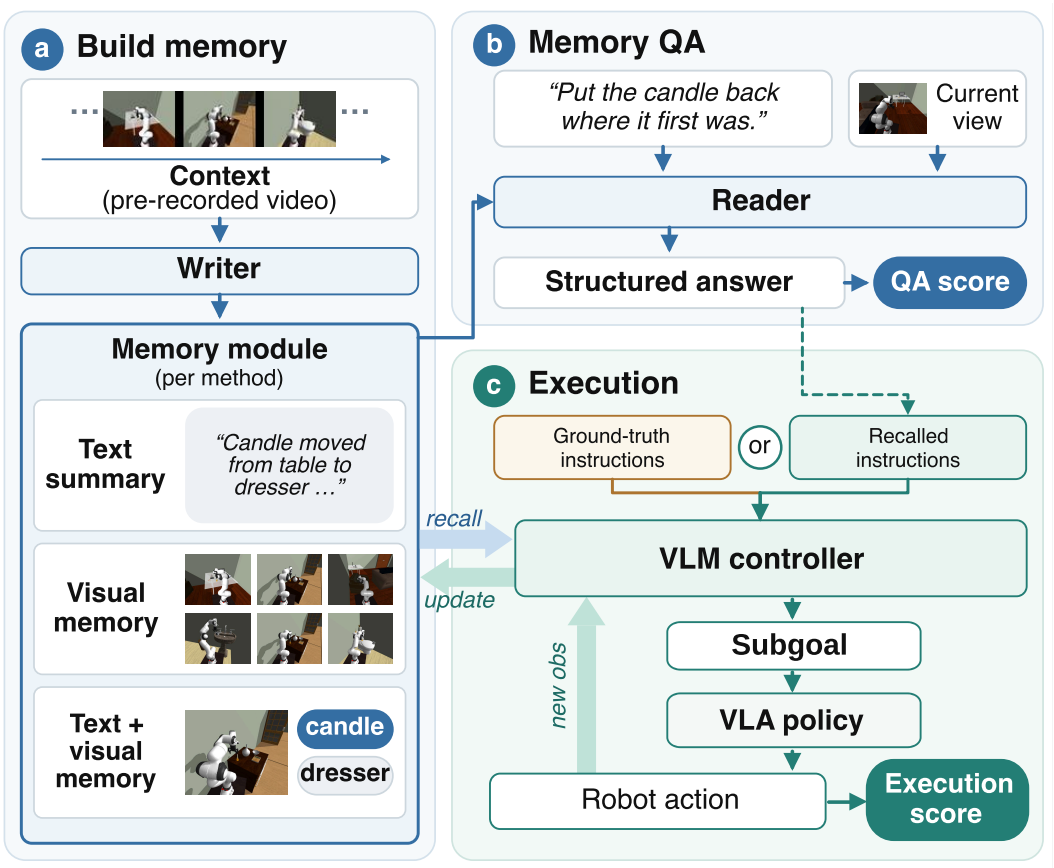

Figure 7: Evaluation harness. (a) Recorded observations are compressed into the memory representation used by each method. (b) A reader combines memory with the request and current view to produce a structured answer. Memory QA evaluates this answer without taking actions. (c) Execution starts from either a supplied correct goal or a goal recovered by QA. The controller updates working memory and chooses a subgoal, then observes the outcome of navigation or manipulation before deciding again. Execution is evaluated on achieved task goals. The memory-update rule varies by method.

Method	Memory update during execution
NO MEMORY	No persistent experience memory is written.
GT ACTION LOG	The supplied context action record remains fixed. No learned online record is added.
UNIFORM FRAMES	Resample frames uniformly from the available observation history.
MEM^{R}	Replace the old text summary with a complete updated summary.
MEMER^{R}	Select a new frame to retain and replace an older one when needed.
R&R AGENT	Add a new text note, retain or replace visual evidence, and select relevant notes under the read budget.

Table 7: How experience memory changes between decisions in the main execution interface. All methods retain access to the supplied goal and current observations.

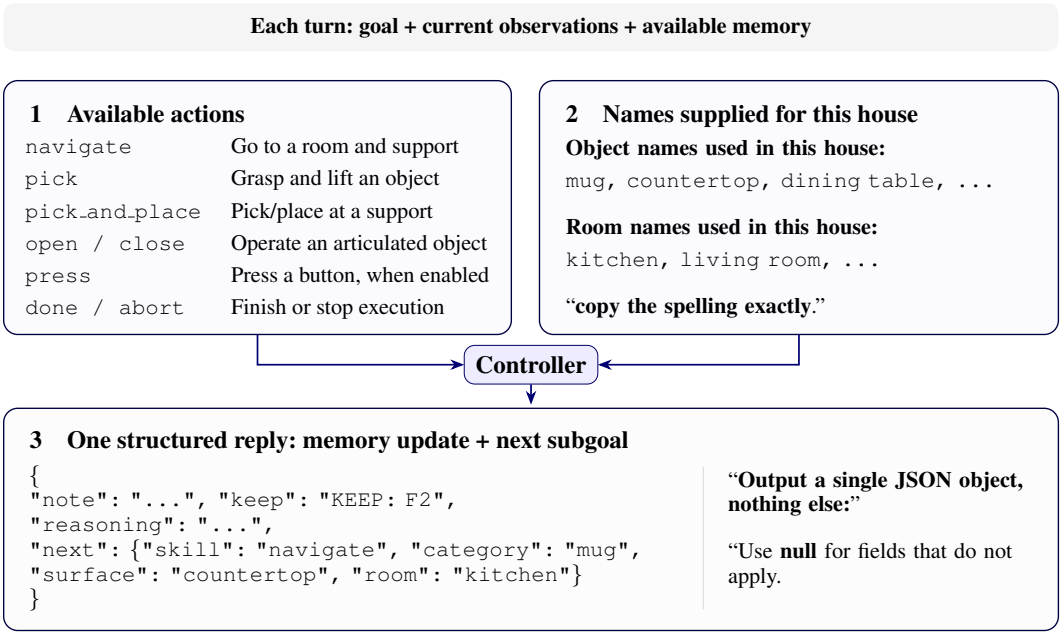

Figure 8: Execution interface provided by the harness. The prompt defines the available skills, supplies object and room names, and specifies the response format. Memory-update fields depend on the method. The returned memory fields update the agent's memory, and `next` selects the subgoal to execute.

B.3 Evaluation metrics and scoring

Memory QA accuracy. An answer is correct only when all required fields match the reference. Finding and Routine compare the room–surface locations for every answer. Restore requires both the current and original room–surface locations to match. Resume requires the exact set of unfinished tasks and the remaining action count for each. An abstention or invalid answer receives no credit.

Execution success rate (SR). A request succeeds when its environmental goal is achieved within the action budget, regardless of whether the agent says it is finished. Finding requires picking the target, or reaching it for requests that only require navigation. Multi requests require every target, whereas Single requests require one valid target. Restore requires placing the object back on its original support. Resume requires completing the remaining actions, and Routine requires placing objects according to the learned rule.

Action-sequence scoring. Before computing ED and Ins., a block of consecutive failed attempts on the same object counts once and is removed if it is eventually recovered. An immediate repeat of the same successful action is treated as a perception retry, navigation counts once per distinct arrival place, and unrecovered failure blocks remain. Ins. counts actions on already-completed work, extra places visited, and unrecovered attempts on non-target objects. For Finding, ED and Ins. use Multi queries only, since a Single query needs only one pick.

All metrics are averaged over the valid episodes of each setting. The number of episodes therefore differs slightly across settings (by at most five), as a few episodes could not be instantiated or replayed reliably in the simulator and were excluded. In addition, Routine tasks requiring five or more subgoals are excluded from execution evaluation, but retained for memory QA.

B.4 How R&R Agent maintains memory

Record. As experience arrives, the writer adds short notes about objects, actions, and places, with time and descriptive tags. It also retains useful frames. Earlier notes stay available, while the retained frames can change under the image budget.

Method	Current correct	Reference correct	Both correct
GT ACTION LOG	52.7	50.3	46.3
NO MEMORY	6.3	1.6	0.3
UNIFORM FRAMES	28.7	24.5	9.6
MEM^{R}	28.7	27.7	11.5
MEMER^{R}	40.9	30.4	16.0
R&R AGENT	**41.7**	**35.0**	**19.0**

Table 8: Restore component accuracy (%) with Gemini 3.7 Flash on 575 queries. Both correct exactly matches the Restore headline accuracy in Table 2. Current and reference denote the present and previous locations. Both must be correct for a successful Restore answer. Bold marks the best memory method in each column, excluding the GT Action Log reference.

Retrieve. The stored tags form an index over notes and frames. When a request arrives, the system matches its object tags to this index. If there is no direct match, the VLM selects a related object category from the available tag list. The matching notes and frames are then retrieved and given to the VLM as memory alongside the request and current view. During execution, the controller uses retrieved memory to choose the next subgoal.

Update. After the VLA executes each subgoal, the controller observes the outcome, adds a note about what it attempted and what visibly changed, and may retain a new frame. The next decision uses the updated memory.

C ADDITIONAL EXPERIMENTAL RESULTS

C.1 MEMORY REPRESENTATIONS ACROSS INFORMATION REQUIREMENTS

Table 2 shows that different requests require different kinds of evidence. On Finding, UNIFORM FRAMES and MEMER^{R} outperform MEM^{R} on Single and Multi Observed queries across all three readers, consistent with the value of retaining visual details that summaries may omit. On Resume, MEM^{R} outperforms MEMER^{R} on both Single and Multi queries across all readers, supporting the usefulness of explicit progress records. On Routine, MEM^{R} also exceeds MEMER^{R} on Manipulated queries, but this advantage does not consistently extend to Observed queries.

R&R AGENT achieves the highest task-average accuracy among the evaluated experience-based memory methods in all 12 task–reader combinations. These results motivate inspecting how memories connect the evidence needed by a request. They do not by themselves establish that combining text and images causes the improvement.

C.2 RECOVERING BOTH LOCATIONS IN RESTORE

Restore asks for both an object's current location and its earlier location. The main QA result counts an answer as correct only when both are right. Table 8 separates these two components on the same queries used in the main comparison.

Among methods that construct memory from experience, R&R AGENT performs best on each component. It identifies the current location in 41.7% of queries and the earlier location in 35.0%, but gets both right in only 19.0%. The differences from the 19.0% joint score correspond to 131 queries with only the current location correct and 92 with only the earlier location correct. Restore therefore requires more than finding one plausible location: the answer must recover both locations and assign each to the correct point in the object's history.

C.3 VISUAL TRACES OF MEMORY USE

Figures 9 and 10 show Finding and Resume memory examples, using Gemini 3.7 Flash under the same evaluation conditions as the main QA results. Finding shows how a wrong destination in a written summary propagates to the answer. Resume shows how an explicit action count supports remaining-work estimation. Each figure places the episode context and instruction above the meth-

Table 9: Selected excerpts from memories written before the question. The same query is used within each row. MEM keeps a rewritten summary. R&R stores separate tagged entries. Quoted fragments are diagnostic selections, not the complete memory.

Task	MEM^{R} summary excerpt	R&R AGENT stored entry excerpt
Finding	"I returned to the bedroom desk, picked up the pen, and placed it on the sofa."	"The robot placed the pen on the bed."
Restore	"I picked up a butter knife from the armchair in the living room and placed it onto the dining table."	"Robot placed the knife on the dining table"
Resume	"I then exited the bedroom into the kitchen and interacted with the microwaveoven on a white table."	"Robot pressed a button on the microwave oven." (count tag: 1)
Routine	"I have placed an egg, an irish potato, a tomato, and an apple onto the living room side table."	"Put the food from the countertop on the side table."

ods' stored evidence, reader reasoning, and answers. Memory excerpts and reasoning are shortened for readability. Stored-image panels display frames at the corresponding entry timestamps.

C.4 EXAMPLES OF STORED MEMORY

Table 9 compares excerpts from the memories saved before the same question under MEM^{R} and R&R AGENT for each task family. MEM^{R} keeps a rewritten summary, while R&R AGENT stores separate tagged entries. These are selected excerpts, not complete memories or final answers.

C.5 VLM SELECTION OF GOAL IMAGES FROM MEMORY

Goal-image baselines. Max-area frame is a recorded image. Using the ground-truth interval when the target was at its original location, we select the external-camera frame in which the target occupies the most pixels. **Target render** is synthesized at the placement hand-off. We temporarily move the held object to its ground-truth original pose in the current simulator scene, render the desired scene through the placement policy's external camera, and restore the object before the policy acts. The camera is randomly positioned for the trial, but the render is not a retrieved memory frame.

Table 6 shows similar placement accuracy for R&R AGENT's selected images and the Max-area frame . Figure 11 shows how their image choices differ. Given the current policy view and eight retrieved frames, the VLM tries to choose an image that shows the object's original position from a similar viewpoint. In the mug and pen examples, the selected frame shows the placement surface and its surroundings much as the robot sees them now. The Max-area frame makes the object larger but changes the view of the scene. This more comparable visual goal may help the policy place the object accurately, even when the target itself appears smaller.

C.6 MEMORY IMAGE BUDGET

Table 10 compares four and eight images on the same Finding and Restore questions. Reducing the budget lowers Finding accuracy from 35.7% to 21.6% for UNIFORM FRAMES and from 44.1% to 29.8% for MEMER^{R}. R&R AGENT changes from 46.6% to 45.0%. On Restore, the corresponding rates are 9.6% to 3.8%, 16.0% to 7.3%, and 19.0% to 17.2%. The smaller change for R&R AGENT is consistent with selecting relevant images from its stored memory.

C.7 RECOVERY AFTER INCORRECT RECALL

An incorrect Memory QA answer rarely leads to successful execution. Among the evaluated runs with an incorrect answer, about 6% of Finding runs recovered by searching elsewhere, about 2% of Restore runs succeeded, and no Resume or Routine runs succeeded.

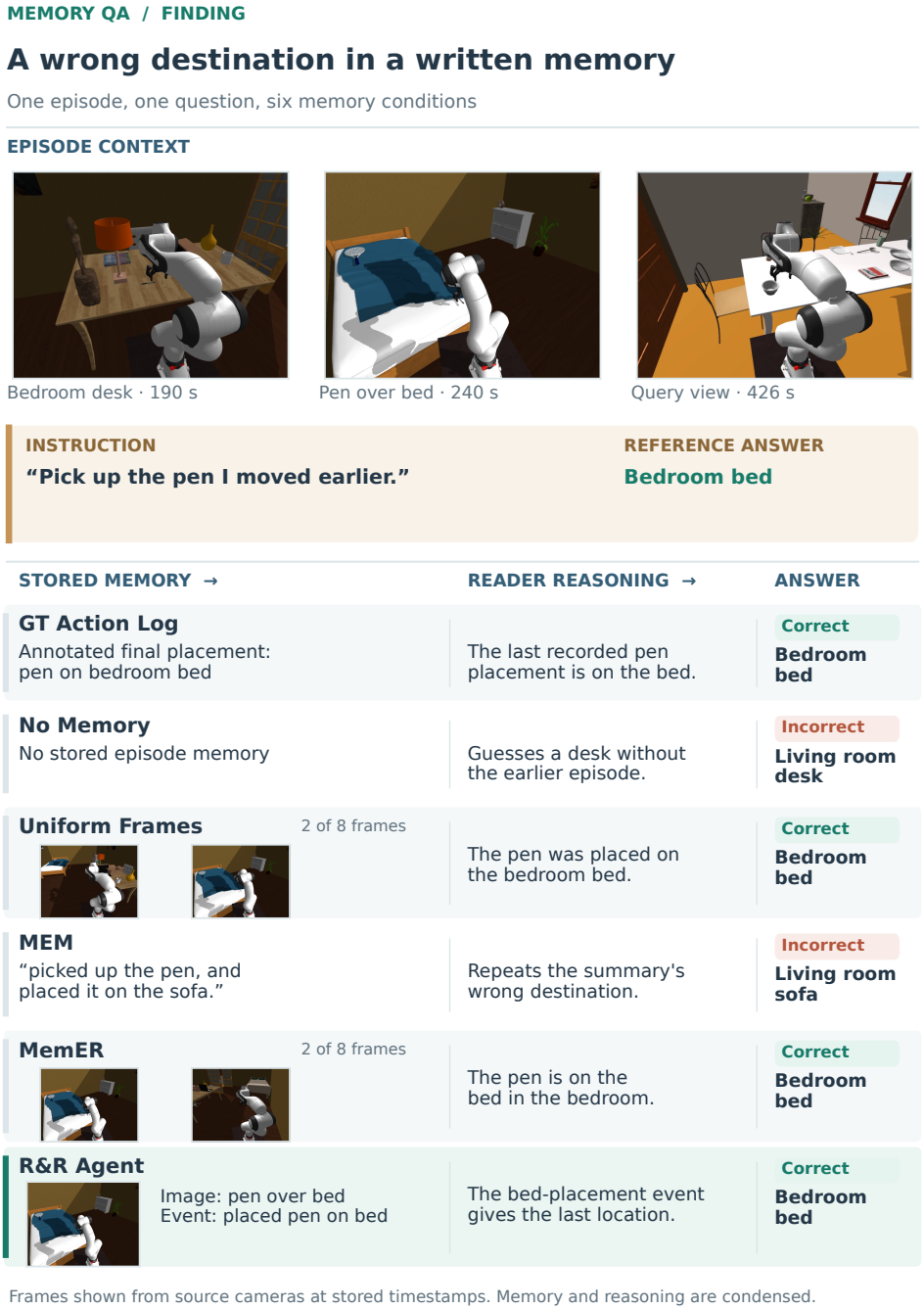

Figure 9: Finding memory example. Both methods identify the pen. MEM^{R} writes the wrong destination, sofa, and its reader repeats it. R&R AGENT retrieves the bed-placement event and a corresponding image, then answers bedroom bed. The GT Action Log is a privileged reference.

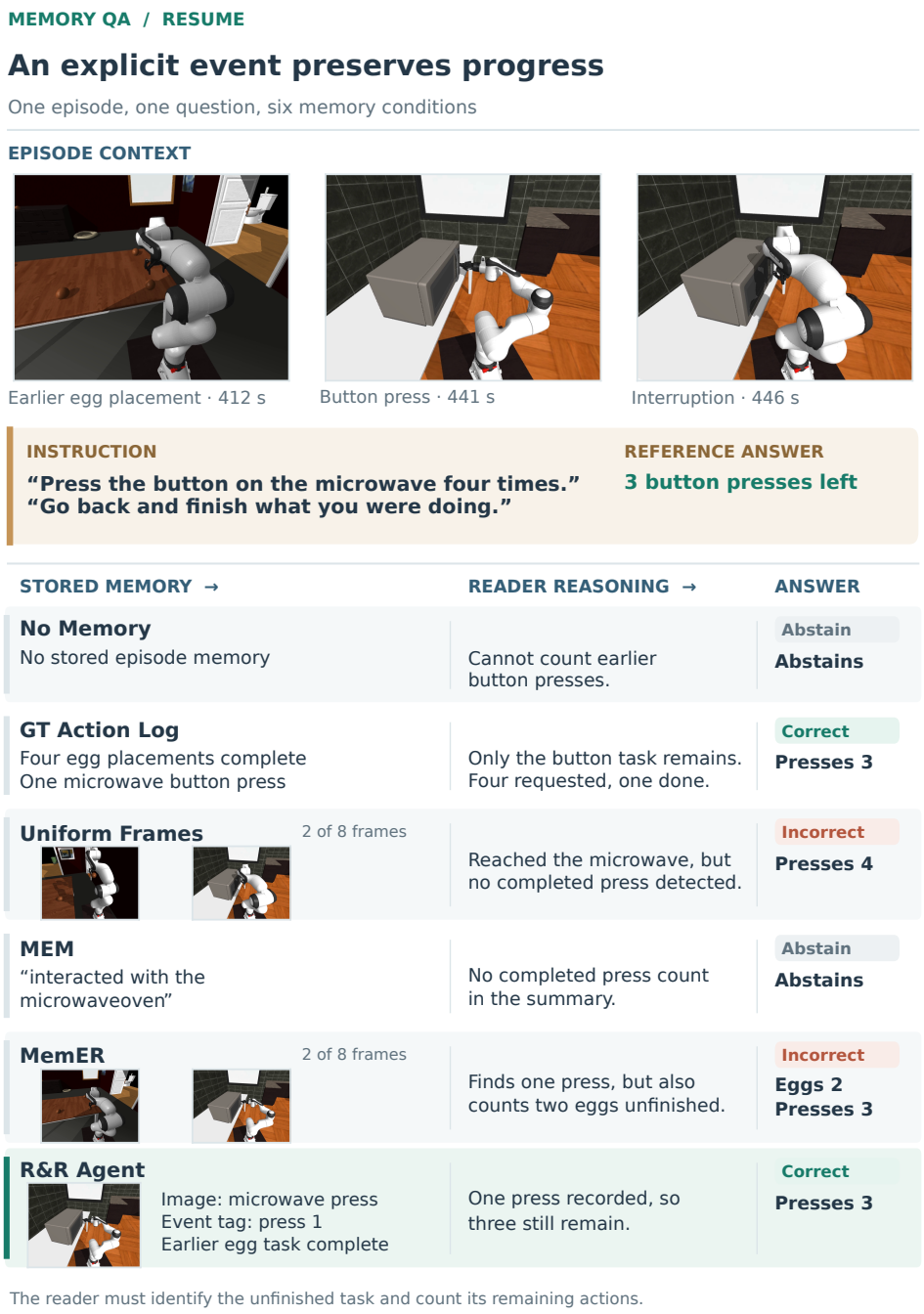

Figure 10: Resume memory example. An earlier egg-moving task is complete, and one of four microwave-button presses has been made. MEM^{R} summarizes the last action only as an interaction and abstains. R&R AGENT retrieves the first press as a tagged event with a corresponding image, then correctly reports three presses remaining. MEMER^{R} recognizes the press but incorrectly keeps part of the egg task unfinished.

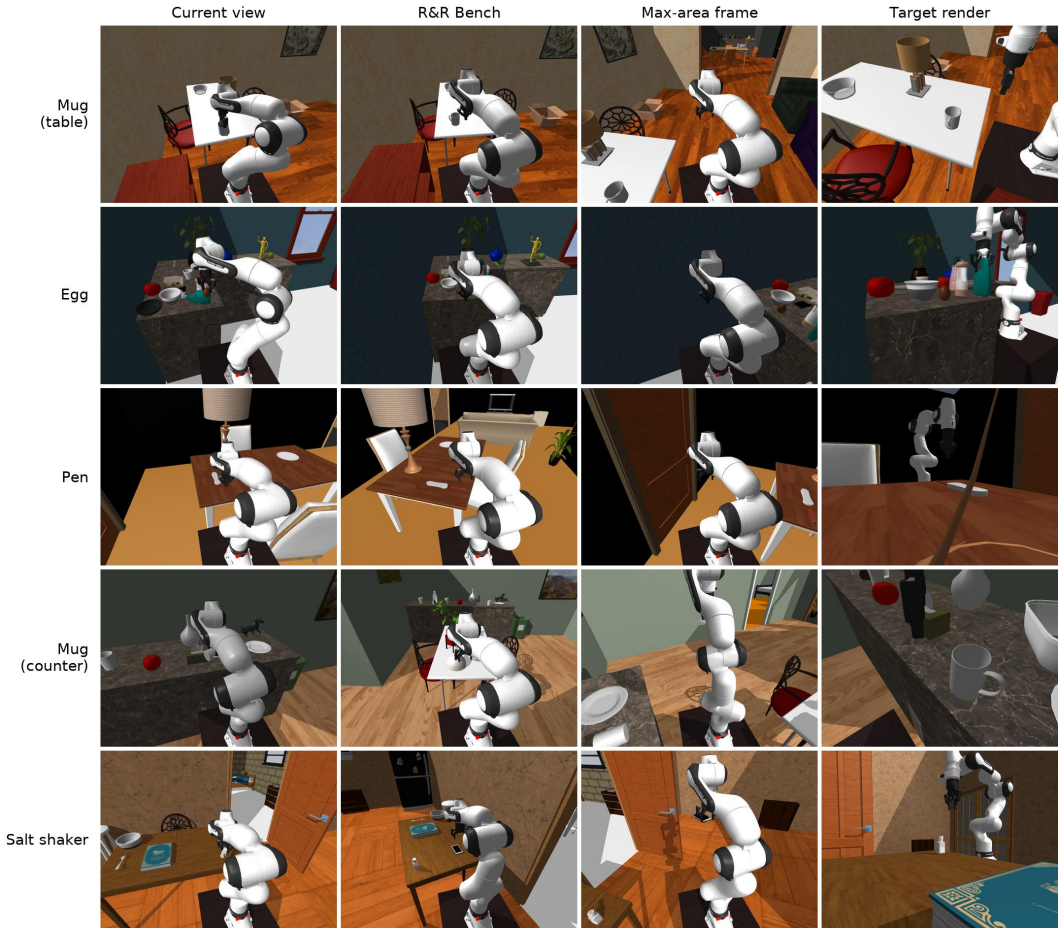

Figure 11: Goal-image candidates for Restore. Each row compares the current scene view with the image selected by R&R AGENT, the past frame with the largest visible target, and a render of the target at its original pose. The selected image can preserve a scene layout closer to the current view even when the target appears smaller. The render is synthesized from the target pose rather than retrieved from memory.

Table 10: Memory QA accuracy (%) with eight or four memory images. All conditions use the same evaluated questions and scoring as the main comparison. Bold marks the best score in each column.

	Finding		Restore	
Method	8 images	4 images	8 images	4 images
UNIFORM FRAMES	35.7	21.6	9.6	3.8
MEMER^{R}	44.1	29.8	16.0	7.3
R&R AGENT	**46.6**	**45.0**	**19.0**	**17.2**

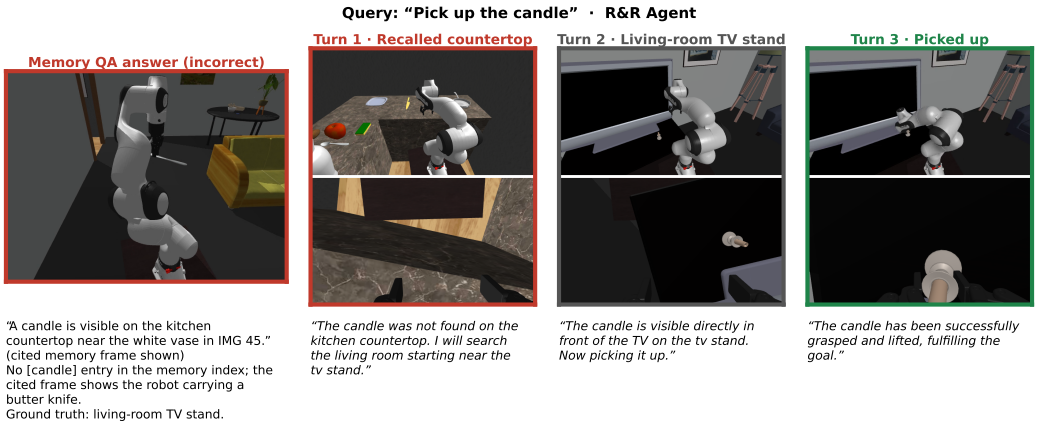

Figure 12: Online recovery after an incorrect memory-QA answer (R&R AGENT, Finding). Left: the wrong answer and cited memory frame. Right: the external and wrist views observed at each subsequent subgoal, with the agent's reasoning.

Figure 12 shows one Finding exception. The reader mistook a memory frame for evidence that the candle was on the kitchen countertop. After finding no candle there, the agent searched the living room and picked it up from the TV stand.

D PROMPTS FORMAT

Selected prompt excerpts. The prompts tell the model what to remember, how to update that memory, and how to return its next action. The boxes below contain selected actual instructions. Bold emphasis is added for readability.

1. What should the robot remember?

Before the request: preserve experience. "**Keep a running account** of what the robot does and what it sees in this house" ... "**You will not see these frames again.**"

During execution: record observed progress.

> "**Do NOT claim a subgoal completed** unless the last frame clearly shows it."
>
> "Prefix anything you are not certain about with **'uncertain: '**."
>
> "Record **what you attempted**, **what visibly changed**, and **what is still left to do**."

2. How does each method update its memory?

MEM^{R} — replace the summary. "Write your **COMPLETE updated notes**."

MEMER^{R} — retain or replace frames. The model selects frames containing useful evidence. The prompt specifies the replacement format: **KEEP:** `F2` **DROP:** `m5`. Here, F2 is a new observation and m5 is a previously stored frame.

R&R AGENT — add a state note and select frames. "Write **ONE new note line** for this turn. Your earlier lines are **kept automatically** — do not repeat them." The newest note remains visible as the current state. Frame selection uses the instruction above.

E VLA FINE-TUNING AND RESULTS

We use MolmoBot-DROID (Deshpande et al., 2026) fine-tuned on manipulation demonstrations as the execution policy, referred to below as *fine-tuned MolmoBot*. It receives current visual observations and a language instruction, without a reference goal image. This section documents its training and separate manipulation diagnostics. These scores are not end-to-end success rates on R&R-Bench.

E.1 TRAINING CONFIGURATION

We fine-tune for 10,000 steps with a global batch size of 256 and a per-device microbatch size of 16 on eight B200 GPUs. Training uses BF16 mixed precision. The policy observes two timesteps separated by eight frames, with one external and one wrist-camera image at each timestep, and predicts Franka joint-position actions. The vision encoder and vision–language connector remain frozen. The language model and action expert are trainable. AdamW (Loshchilov & Hutter, 2017) uses learning rates of 10^{-5} for the language model and 10^{-4} for the action expert, with 2,000 and 200 warm-up steps, respectively.

The sampling mixture combines MolmoBot-Data pick-and-place part 1 (24%), pick (12%), pick-and-place part 0 (8%), next-to placement (12%), and color-conditioned placement (8%) with additional placement (16%), opening (10%), and closing (10%) demonstrations. The latter three retain their recorded instructions, including *"Place the X on the [support]."* These last three datasets extend the original MolmoBot training mixture. The place-only clips begin with an object already grasped and teach placement and release. The Open and Close clips add drawer manipulation demonstrations. We also train a separate variant with button-press demonstrations on coffee makers, microwaves, washing machines, and dryers. Fine-tuned MolmoBot uses no reference goal images during training or inference. Section E.3 describes goal-image conditioning and its effect on placement.

E.2 MANIPULATION ABILITIES AFTER FINE-TUNING

Table 11 first compares MolmoBot before and after fine-tuning, then compares fine-tuned MolmoBot with $\pi_{0.5}$. The policies are tested on the same scenes for each Pick or Place comparison. Place starts with the object already grasped.

Pick and Place are common in our benchmark, so we use fine-tuned MolmoBot as the execution policy. In these skill tests, it outperforms fine-tuned $\pi_{0.5}$ (Black et al., 2025) on Pick (67.5% vs. 45.0%) and on placing an object within 10 cm of the target (17.5% vs. 6.8%). $\pi_{0.5}$ performs better on Open and Close.

E.3 GOAL-IMAGE CONDITIONING

We additionally train a goal-image-conditioned MolmoBot policy to specify the desired placement when the instruction names only the support. Following $\pi_{0.7}$ (Physical Intelligence et al., 2026), we append the goal image last. Its tokens can attend to earlier inputs, while the earlier tokens cannot attend to the goal image.

The goal image is the final external-camera frame of the demonstration. We provide it in two-thirds of training examples. The goal-conditioned and language-only policies use the same demonstration mixture and 10,000 training updates.

We test 130 shared states in which the robot already holds the object, isolating placement. The target is the object's final position after an expert places it, as shown in the correct goal image. The instruction names the support surface but not this exact position. We compare the language-only policy with the goal-image policy given the correct image, no image, or an incorrect image.

Table 11: Language-only manipulation success (%). The first panel compares MolmoBot before and after fine-tuning. The second compares fine-tuned MolmoBot and $\pi_{0.5}$.

MolmoBot fine-tuning		
Measure	Pretrained	Fine-tuned
Place within 10 cm	11.5	17.7
Next-to success	41.5	45.4
Fine-tuned policy comparison		
Measure	MolmoBot	$\pi_{0.5}$
Pick success	**67.5**	45.0
Pick and retain grasp	**35.0**	22.5
Place within 10 cm	17.5	6.8
Open success	8.0	11.0
Close success	42.0	56.5

Replace@r means releasing the object within r centimeters horizontally and 5 cm vertically of the target.

With the correct image, Replace@10 cm rises from 13.8% to 63.8% for the same goal-image policy (Table 12). An incorrect image lowers success to 7.7%, indicating that the policy uses the image content. The language-only policy reaches 17.7%.

Table 12: Goal-image placement on 130 matched hand-off states. Success rates are percentages. Median error is in centimeters and lower is better.

Policy	Reference image	@5 cm	@10 cm	@15 cm	Median error
Fine-tuned MolmoBot	None	5.4	17.7	25.4	23.6
Goal-image-conditioned MolmoBot	Withheld	4.6	13.8	22.3	25.6
Goal-image-conditioned MolmoBot	Incorrect	2.3	7.7	12.3	43.2
Goal-image-conditioned MolmoBot (ours)	Supplied	**35.4**	**63.8**	**75.4**	**6.7**