SIMPLE AGENTIC MEMORY FOR GENERALIST ROBOT POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Visual-memory systems commonly retain or compress past observations. Robot control additionally requires interaction-derived state that no individual frame may explicitly represent, such as persistent identity relations, accumulated progress, or ordered procedures. We introduce *Simple Agentic Robot Memory* (**SimpleARM**), a training-free memory layer for frozen generalist robot policies. From the task instruction, SimpleARM specifies what to monitor; frozen perceptual tools maintain compact typed state online; structured access retrieves that state only when a proposed subgoal depends on history; and current-view grounding resolves recalled entities before execution. We evaluate SimpleARM on RoboMME, a benchmark of memory-dependent robot manipulation tasks that require history information no longer available in the current observation. Across all 16 tasks and three policy seeds, SimpleARM achieves **67.17%** mean success, compared with 44.51% for the strongest non-oracle baseline. Matched ablations show mechanism specificity: removing relation, reference, progress, or route state produces large losses where the affected state is retrieved for control, while largely sparing other tasks. These results support a state-based view of robot memory: effective memory for control is not simply retained visual history, but compact task-relevant state derived from the interaction history.

1 INTRODUCTION

Vision–language–action (VLA) models are expanding robot policies from isolated skills toward generalist control over diverse objects, embodiments, and language instructions (Brohan et al., 2023; Kim et al., 2025; Black et al., 2025b). As instructions become more compositional and tasks span longer horizons, success depends on more than interpreting the current view: an embodied agent must remember what it observed, what its own actions changed, and what remains to be done. Memory is therefore a core capability for embodied agents, just as context management and experience accumulation are for language and multimodal agents (Packer et al., 2023; Wang et al., 2023; Fan et al., 2025).

Long-video understanding offers increasingly capable ways to sample, compress, and retrieve visual history, while streaming-video models make this process causal and memory-bounded (Wang et al., 2024; Fan et al., 2024; Zhang et al., 2025; Chen et al., 2024; Huang et al., 2025; Liang et al., 2026). In both settings, history is commonly represented first as an observation sequence, and memory is organized around preserving useful visual evidence from that sequence. Yet RoboMME's own memory-budget study shows increasingly modest gains from allocating more tokens to perceptual history (Dai et al., 2026), suggesting that capacity alone does not resolve the memory problem. Robot control instead requires asking not only which observations to retain, but which task-relevant state to maintain. Its object of memory is an interleaved history of observations and actions: actions change both the world and what the agent will observe next, so under partial observability the state sufficient for control must summarize past actions as well as observations (Kaelbling et al., 1998). Crucially, the information needed for the next decision may not be contained in any single frame. Robot memory is not exhausted by selecting a representative subset of frames. *Robot memory is a problem of maintaining the task-relevant state induced by an action–observation history.*

This formulation has a cognitive analogue: human visual working memory is strongly selective, yet its retained content is not uniformly converted into symbols (Vogel et al., 2005; Harrison &

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.

Tong, 2009; Kwak & Curtis, 2022). This suggests a computational principle: preserve perceptual processing for the current scene while explicitly structuring the identity, progress, and trajectory variables that must persist across time.

RoboMME provides a concrete testbed for this principle through memory-dependent tasks for robotic generalist policies (Dai et al., 2026). Many tasks exhibit *perceptual aliasing*: the same current observation can be compatible with different interaction histories and therefore require different next actions (Chrisman, 1992). The robot may need to remember which container covered an object, which block was manipulated in a demonstration, how many repetitions have been completed, or which route was shown earlier. In each case, the task-relevant state is determined by the interaction history but is not fully recoverable from selected frames alone.

We address this ambiguity with **SimpleARM**, a training-free memory layer around a frozen grounded-subgoal predictor and VLA policy. Before execution, an instruction-conditioned VLM proposes a candidate state specification and compatible retrieval operations. During interaction, frozen perceptual and event tools verify evidence and maintain entity–relation bindings, event progress, and ordered trajectories or procedures. Structured access retrieves only state compatible with a history-dependent field of the proposed subgoal, and current-view perception re-grounds recalled entities before execution. This division of labor defines the agentic design: language specifies *what* may matter, tools estimate and update *what is true*, and structured access determines *when* maintained state can affect control.

Our contributions are threefold:

- We formulate robot memory as task-conditioned maintenance of *policy-sufficient state* over an action–observation history, rather than visual-history compression alone.
- We introduce *Simple Agentic Robot Memory* (**SimpleARM**), which derives a task-specific memory specification, maintains compact state online, and combines structured retrieval with current-view grounding to expose that state to a frozen policy.
- Across all 16 RoboMME tasks, our method achieves **67.17%** mean success over three seeds, exceeding both the no-memory baseline (32.70%) and the strongest previously reported non-oracle method (44.51%). Ablations show that removing relation, reference, progress, or route state causes large losses where that state is retrieved for control, while largely sparing other tasks.

2 Related Work

2.1 Agentic and Online Video Understanding

Long-video agents turn recordings into query-relevant evidence through iterative frame search, visual tools, instruction-conditioned memory, or hierarchical reasoning (Wang et al., 2024; Fan et al., 2024; Diko et al., 2025; Yin et al., 2026; Yeo et al., 2026; Zhou et al., 2025). The central problem is selecting evidence from a fixed visual history. Online video models instead process frames causally and answer temporally aligned or asynchronous queries under bounded memory (Zhang et al., 2025; Chen et al., 2024; Huang et al., 2025; Liang et al., 2026). SimpleStream shows that a sliding window of recent frames can be a strong streaming baseline, motivating an explicit distinction between recent-scene perception and long-range memory (Shen et al., 2026). Recent systems further organize streams into sparse, entity-centric, or embodied scene memories (Jiang et al., 2026; Long et al., 2026; Fan et al., 2025). Robot control shares this causal, bounded-memory setting but couples perception to action: each action changes the scene and subsequent observations. Memory must therefore supply a control-relevant variable and, for recalled entities, re-ground identity in the current view before action rather than return historical evidence only as an answer.

2.2 Memory for VLAs

RoboMME (Dai et al., 2026), MIKASA-Robo (Cherepanov et al., 2026), RMBench (Chen et al., 2026), and RoboMemArena (Lei et al., 2026a) evaluate complementary forms of history dependence and partial observability in robot manipulation. Memory-augmented VLA methods likewise span several representations: MemoryVLA consolidates perceptual and cognitive tokens (Shi

et al., 2026); VPWEM combines working and episodic memory (Lei et al., 2026b); MEM combines short-horizon visual and long-horizon textual memory (Torne et al., 2026); Chameleon learns control-indexed episodic state (Guo et al., 2026); and MemER retrieves task-relevant keyframes for high-level action prediction (Sridhar et al., 2026).

Several systems connect semantic memory to visuomotor control through a hierarchical interface. RoboMME calls its high-level VLM a *subgoal predictor*, and its symbolic-memory variants encode history as language subgoals (Dai et al., 2026). MemER similarly retrieves keyframes to produce text instructions for a low-level VLA (Sridhar et al., 2026). More broadly, $\pi_{0.5}$ uses semantic subtask prediction for hierarchical inference (Black et al., 2025a), while Hi Robot uses a high-level VLM to infer the next step for a low-level VLA (Shi et al., 2025). We follow this terminology and call the semantic planner the *high-level subgoal predictor*.

These methods differ in what history reaches the policy: compressed visual tokens, retrieved frames, or text. RoboMME further reports diminishing returns from scaling perceptual-memory tokens (Dai et al., 2026). **SimpleARM** instead maintains instruction-conditioned entity–relation, progress, and trajectory state, retrieves it only for compatible history-dependent subgoals, and re-grounds recalled entities in the current view. This training-free layer leaves both the high-level subgoal predictor and low-level VLA frozen.

2.3 Selective memory for grounded action

General agent memories study what to store, consolidate, and retrieve for later reasoning (Packer et al., 2023; Xu et al., 2025). Closed-loop control additionally requires state to track action-induced scene changes and retrieval to affect the policy only when the current decision depends on history. Human visual working memory offers a complementary analogy: it selectively admits task-relevant objects (Vogel et al., 2005), retains stimulus-specific content (Harrison & Tong, 2009), recodes content into goal- and action-oriented abstractions (Kwak & Curtis, 2022; Boettcher et al., 2021), and organizes continuous experience into event models (Kurby & Zacks, 2008). Together, these findings motivate preserving current visual processing while explicitly structuring only the identity, progress, and trajectory variables needed for future action.

3 Method

3.1 Policy-Sufficient State

Let g denote the task instruction, d an optional demonstration, o_t the observation at time t, and a_t the executed action. The information available to the agent is the interaction history

$$h_t = (g, d, o_0, a_0, \ldots, a_{t-1}, o_t). \tag{1}$$

SimpleARM maintains an episode-scoped state through a task-conditioned recursive update,

$$M_t = \mathcal{U}(M_{t-1}, o_t, a_{t-1}; g, d), \tag{2}$$

with the objective that the current observation and memory retain the information from h_t needed for control:

$$\pi(a_t \mid h_t, g) \approx \pi(a_t \mid o_t, g, M_t). \tag{3}$$

The policy-sufficient memory state need not correspond to any subset of past observations. Rather than preserving only visual evidence, it can represent derived interaction state: persistent entity–relation bindings, accumulated event progress, and ordered trajectories or procedures.

Figure 1 summarizes **SimpleARM**. Under hierarchical control, the high-level subgoal predictor proposes the next subgoal and a frozen VLA policy executes it. **SimpleARM** augments this pipeline in three stages: it specifies what historical information may matter for future control, thereby restricting memory to task-relevant state; maintains that state online using frozen perception and event tools, thereby preserving reliable information across time; and selectively retrieves it only when the current subgoal depends on interaction history, thereby limiting irrelevant memory influence on control.

Before execution, an instruction-conditioned memory specification identifies candidate state variables to track. During interaction, frozen perceptual and event tools extract evidence from the observation stream and update these variables. At decision time, structured access determines whether the

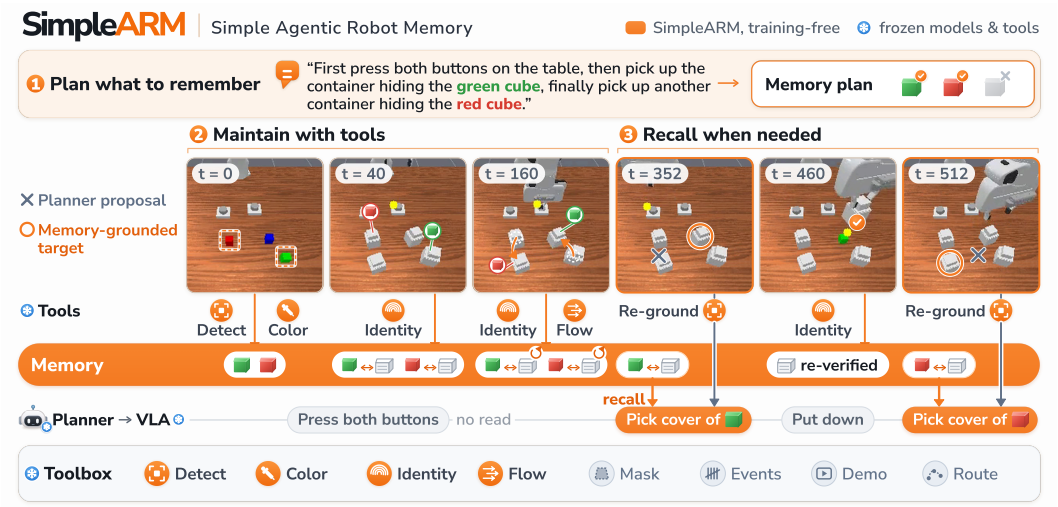


Figure 1: **SimpleARM overview.** Given the instruction, SimpleARM first specifies what historical state may be needed, then maintains that state online with frozen perception and event tools, and retrieves it only when the current subgoal depends on interaction history. Retrieved memory resolves historical identity, progress, or procedure; current perception provides any required spatial grounding, and the resulting memory-conditioned grounded subgoal is executed by the frozen VLA.

predicted subgoal depends on history; retrieved state is verified and, when necessary, re-grounded in the current observation before being passed to the policy. Thus, language specifies *what* may matter, tools estimate and update *what is true*, and structured access determines *when* maintained state can affect control.

Formally,

$$(P, A) = \mathcal{C}(g), \qquad M_t = \mathcal{U}(M_{t-1}, \Phi(o_t, a_{t-1}, d); P)\,, \qquad \hat{u}_k = \mathcal{G}_\theta(g, o_t), \tag{4}$$

$$r_k = \mathcal{R}(\hat{u}_k, M_t; P, A), \quad \tilde{u}_k = \mathcal{V}(\hat{u}_k, r_k, o_t), \quad a_{k:k+H} \sim \pi_{\mathrm{VLA}}(\cdot \mid o_t, \tilde{u}_k). \tag{5}$$

Here, $\mathcal{C}$ produces a candidate memory specification P and compatible reader set A. The tools Φ extract evidence from the current observation, executed action, and optional demonstration. $\mathcal{U}$ maintains the resulting typed state. We instantiate the abstract update in Eq. 2 by first compiling a task-conditioned specification P and extracting observable evidence $\Phi(o_t, a_{t-1}, d)$. $\mathcal{G}_\theta$ is the frozen high-level subgoal predictor. Given its proposed subgoal $\hat{u}_k$, $\mathcal{R}$ reads only compatible historical state, while $\mathcal{V}$ verifies the retrieved information and, when needed, re-grounds referenced entities in the current observation. The resulting subgoal $\tilde{u}_k$ is then executed by the frozen VLA policy π_{VLA}. SimpleARM separates three roles: the task instruction defines a compact state abstraction, perceptual and event tools estimate that state from interaction, and structured access determines when it can influence control.

3.2 Task-Conditioned State Abstraction

A complete interaction history contains redundant and distracting information for a single task. SimpleARM therefore constructs a task-conditioned abstraction of that history: the instruction specifies a small set of state variables that may be needed for future decisions, and perceptual and event tools estimate those variables as interaction unfolds. Unlike a fixed latent representation, the state space itself is configured by the task.

Task-conditioned state specification. Before execution, a task-conditioned memory specification produces a candidate state specification

$$P_g = \mathcal{C}(g) = \left(P_g^{\mathrm{ent}}, P_g^{\mathrm{prog}}, P_g^{\mathrm{traj}}\right), \tag{6}$$

describing which persistent relations, progress variables, and ordered procedures may be relevant to the task. The specification defines *what should be estimated*, rather than its value: language proposes the state variables, while their values are established from observations, actions, and an optional demonstration by frozen perceptual and event tools.

The resulting episode state is

$$M_t = \left(M_t^{\text{ent}}, M_t^{\text{prog}}, M_t^{\text{traj}}\right), \tag{7}$$

where the factors capture complementary forms of history-dependent information. M_t^{ent} is an object-centric relational state that preserves identities and task-relevant relations after their original visual evidence becomes unavailable; M_t^{prog} represents task progress accumulated through interaction, such as event counts or repetition state; and M_t^{traj} represents ordered procedural state, such as a demonstrated route or sequence together with its current progress. These variables summarize information induced by the interaction history rather than retaining the observations themselves. Their representation and update rules differ, but all are instantiated only when selected by the task-conditioned memory specification. This factorization is not intended as a complete world model: it retains only variables selected as potentially relevant to future control.

Tool-grounded state updates. Each maintained variable is updated from evidence produced by an appropriate perceptual or event tool. Let

$$z_t^i = \Phi_i(o_t, a_{t-1}, d) \tag{8}$$

denote candidate evidence for state variable m^i. An update is committed only when the corresponding evidence gate accepts that evidence:

$$m_t^i = \begin{cases} \mathcal{U}_i(m_{t-1}^i, z_t^i), & \mathcal{A}_i(z_t^i, o_t) = 1, \\ m_{t-1}^i, & \text{otherwise.} \end{cases} \tag{9}$$

Here, Φ_i extracts evidence relevant to the maintained variable, while $\mathcal{A}_i$ determines whether that evidence is sufficient to update persistent state. Thus, the task-conditioned specification determines *what* should be maintained, while tool-grounded evidence determines *when and how* that state changes. Details are described in Appendices A.2 and A.3.

3.3 Selective Memory Retrieval and Current-View Grounding

The task-conditioned state in Sec. 3.2 may contain information that is relevant somewhere in the episode but unnecessary for the current decision. SimpleARM therefore retrieves memory selectively rather than exposing the full maintained state to the policy. At decision step k, the frozen high-level subgoal predictor proposes

$$\hat{u}_k = \mathcal{G}_\theta(g, o_t). \tag{10}$$

The instruction-conditioned retrieval specification A then determines whether the proposed subgoal contains a history-dependent argument that can be resolved from M_t. Let $\mathcal{Q}$ construct the corresponding memory query:

$$q_k = \mathcal{Q}(\hat{u}_k; P, A), \qquad r_k = \begin{cases} \mathcal{R}(M_t, q_k), & q_k \neq \varnothing, \\ \varnothing, & q_k = \varnothing. \end{cases} \tag{11}$$

Thus, memory retrieval is conditioned jointly on the task specification and the current subgoal. An entity or relation query retrieves the historical referent needed to resolve an object argument; a progress query retrieves an accumulated count, ordinal, or task stage; and a procedural query retrieves the relevant element of a maintained sequence. A retrieval operation proposed from the instruction may therefore never fire if no compatible history-dependent argument appears during execution.

Historical state may determine *which* entity is relevant without specifying *where* it is in the current observation. We therefore separate memory retrieval from current-view grounding:

$$\gamma_k = \begin{cases} \text{Ground}(r_k, o_t), & \text{if current-view localization is required,} \\ \varnothing, & \text{otherwise,} \end{cases} \qquad \tilde{u}_k = \mathcal{V}(\hat{u}_k, r_k, \gamma_k). \tag{12}$$

Entity and relation state may require current-view grounding, whereas progress and procedural state can directly refine symbolic subgoal arguments.

Memory modifies the proposal only when compatible historical state is available and any required current grounding succeeds; otherwise, $\tilde{u}_k = \hat{u}_k$. Together with Sec. 3.2, this yields a simple pipeline: the instruction specifies what to maintain, online evidence updates that state, and selective retrieval determines when it influences the current subgoal. Appendix E.6 illustrates this process over a complete episode.

3.4 Training-Free Integration

SimpleARM introduces a training-free memory layer for maintaining and selectively retrieving task-relevant state, while keeping the pretrained subgoal predictor and VLA policy frozen. The high-level subgoal predictor is RoboMME's fine-tuned Qwen3-VL-4B-Instruct model (Bai et al., 2025), which predicts grounded subgoals from visual observations, while the low-level controller is its $\pi_{0.5}$-based GroundSG VLA policy (Dai et al., 2026; Black et al., 2025a). Both remain frozen. The memory layer uses frozen Qwen3-VL-4B-Instruct as an instruction-only planner to derive two specifications before control begins. The state specification P identifies candidate entity relations, progress signals, and procedures to maintain; the retrieval specification A describes how those records may resolve a future history-dependent subgoal. This planning step proposes the memory schema but neither assigns state values nor decides at runtime when memory is read. Frozen perception and event tools maintain the state from observations, structural access retrieves compatible records, and current-view perception grounds recalled entities before execution. The query structure and vote aggregation are described in Appendix A.1. The added memory is non-parametric and episode-scoped: M_0 is empty, state evolves only through the updates in Sec. 3.2, and all records are reset between episodes. Appendix A gives the complete RoboMME instantiation.

4 Experiments

Our experiments ask three questions that follow the state-to-action decomposition of Section 3. **RQ1** asks *when* maintaining task state improves over retaining visual history, comparing SimpleARM with published and episode-matched retained-frame baselines at the aggregate, suite, and episode level. **RQ2** asks *which* components matter: matched ablations test whether different memory demands rely on different maintained variables, and whether a variable helps only when structured access retrieves it for a decision. **RQ3** asks *what remains* beyond memory, examining representative failures in which the outcome hinges on state correctness, identity ambiguity, temporal perception, or downstream control.

4.1 Experimental Setup

We evaluate all sixteen RoboMME tasks (Dai et al., 2026), grouped into the official *Counting*, *Permanence*, *Reference*, and *Imitation* suites, across three seeds. We initialize the high-level subgoal predictor from RoboMME's fine-tuned Qwen3-VL-4B-Instruct grounded-subgoal adapter and the VLA from its $\pi_{0.5}$-based GroundSG policy. Both are used without further training; the perception models also remain frozen, and the added memory receives no training or test-time optimization. Baselines are RoboMME's published three-seed $\pi_{0.5}$ (no-memory) and FrameSamp-Modul results and its non-deployable oracle-subgoal ceiling (Dai et al., 2026), together with RoboMME's evaluation of MemER (Sridhar et al., 2026; Dai et al., 2026). Motivated by SimpleStream's recent-frame baseline for streaming video (Shen et al., 2026), we also report Recent32+ModuL, RoboMME's FrameSamp+ModuL architecture trained and evaluated with the latest 32 frames as memory, using the same three policy seeds as our method (50 episodes per task per seed); the Recent4 and Recent16 variants and full protocol details are given in Appendix B.

4.2 RQ1: When Does State Maintenance Improve over Visual-History Retention?

SimpleARM achieves **67.17%** average success (Table 1), outperforming the strongest baseline FrameSamp result by 22.66 points and $\pi_{0.5}$ without memory by 34.5 points. Recent32+ModuL

Table 1: **Success rate (%) on RoboMME.** All columns use three policy seeds and 50 episodes per task per seed. Published baseline and oracle results are from RoboMME (Dai et al., 2026), which also evaluates MemER (Sridhar et al., 2026); Recent32+ModuL retains the latest 32 native frames (Sec. 4.1). Δ_{FS} is ours minus FrameSamp. State labels are not mutually exclusive. Bold and underlining mark the best and second-best non-oracle results, respectively; ties share the same formatting. The ground-truth-subgoal oracle is shown in gray.

			success (%)					
suite	task	required state	no mem.	MemER	FrameSamp	Recent32	ours	oracle
Counting	BinFill	progress	52.0	56.7	39.6	34.0	**60.0**	85.8
	PickXtimes	progress	**92.7**	79.3	87.3	83.3	91.3	100.0
	SwingXtimes	progress	7.3	59.3	**92.0**	80.7	76.7	100.0
	StopCube	progress	0.0	0.0	**42.0**	41.3	0.0	49.7
	suite avg.	$\Delta_{\mathrm{FS}} = -8.22$	38.00	48.83	**65.22**	59.83	57.00	83.86
Permanence	VideoUnmask	relation	88.7	81.3	32.7	19.3	**96.0**	98.8
	ButtonUnmask	relation	24.0	72.0	25.1	20.0	**82.0**	95.0
	VideoUnmaskSwap	relation	30.7	38.0	24.4	23.3	**80.0**	99.2
	ButtonUnmaskSwap	relation+progress	14.0	21.3	18.2	28.0	**56.7**	80.2
	suite avg.	$\Delta_{\mathrm{FS}} = +53.56$	39.34	53.17	25.11	22.67	**78.67**	93.31
Reference	PickHighlight	relation+progress	15.3	**70.7**	22.9	26.7	66.0	83.3
	VideoRepick	identity+progress	25.3	25.3	30.4	26.7	**78.0**	97.3
	VideoPlaceButton	identity+order	54.0	30.0	60.0	26.0	**82.0**	100.0
	VideoPlaceOrder	identity+order	31.8	26.0	32.0	24.7	**88.0**	100.0
	suite avg.	$\Delta_{\mathrm{FS}} = +42.17$	31.61	38.00	36.33	26.00	**78.50**	95.17
Imitation	MoveCube	procedure	71.6	**82.7**	77.8	52.0	67.3	87.8
	InsertPeg	procedure	3.3	6.7	**7.6**	2.0	4.0	15.6
	PatternLock	trajectory+order	6.7	16.7	53.6	8.7	**93.3**	97.0
	RouteStick	trajectory+order	6.0	12.0	**66.7**	7.3	53.3	55.6
	suite avg.	$\Delta_{\mathrm{FS}} = +3.11$	21.89	29.50	51.39	17.50	**54.50**	63.98
Overall avg.		$\Delta_{\mathrm{FS}} = +22.66$	32.70	42.37	44.51	31.50	**67.17**	84.08

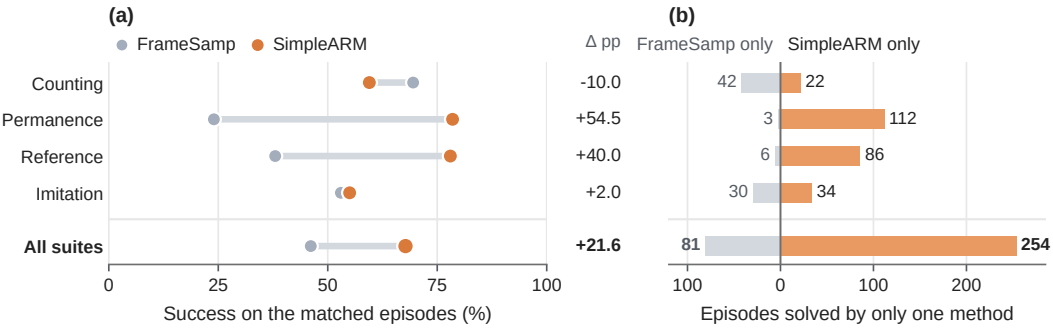


Figure 2: **SimpleARM's paired gains are concentrated in Permanence and Reference.** (a) Success rates of SimpleARM and the released FrameSamp-Modul checkpoint on matched episode identities under the same seed (200 episodes per suite; 800 overall). Δ denotes the paired success-rate difference (SimpleARM minus FrameSamp), in percentage points. (b) Counts of episodes solved by exactly one system: FrameSamp alone (left) or SimpleARM alone (right). The bottom row in each panel pools all four suites.

reaches $31.50 \pm 0.43\%$ over the same three policy seeds, 13.0 points below FrameSamp. The episode-matched comparison shows a similar margin: on the same 800 episodes, our method succeeds on 542, compared with 369 for the released FrameSamp-Modul checkpoint (+21.6 points), with 254 ours-only versus 81 FrameSamp-only successes. Together, these results indicate a substantial system-level advantage over retained-frame memory, although this comparison alone does not isolate which gains are specifically attributable to the memory layer.

SimpleARM delivers its largest gains on *Permanence* and *Reference*, outperforming FrameSamp by 53.56 and 42.17 points, respectively (Table 1). It also improves Imitation by 3.11 points. The only suite-level deficit is Counting (-8.22 points), driven largely by the unsupported StopCube task;

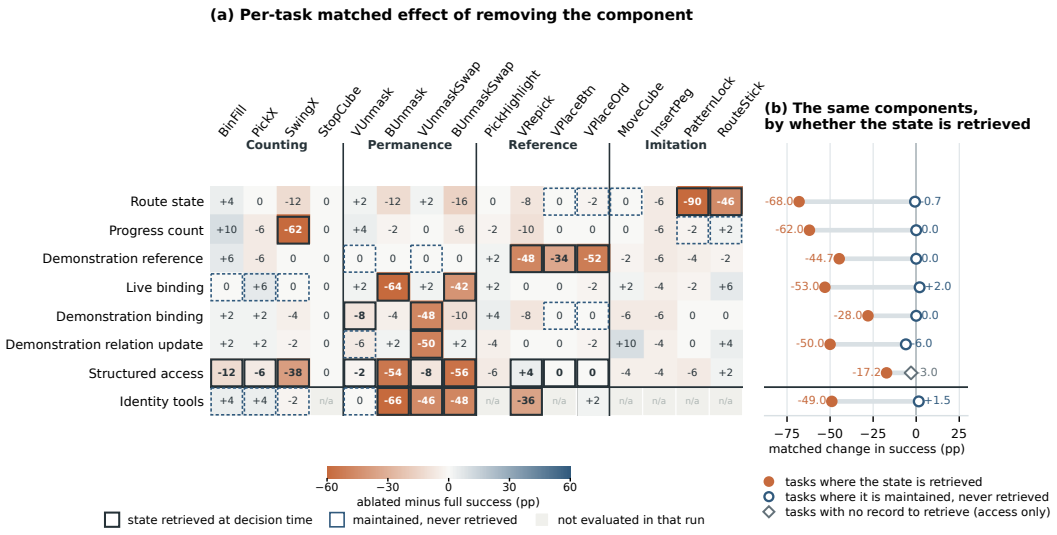


Figure 3: **Matched ablations localize to tasks that retrieve the state.** (a) Cells show the matched change in success after removing one component (ablated minus full; 50 paired episodes per task). Solid borders mark tasks that retrieve the maintained state for control; dashed borders mark tasks where the mechanism maintains state in at least 20% of episodes but it is never retrieved. The identity-tools row uses a separate 450-episode evaluation over nine tasks; unevaluated cells are marked *n/a*. (b) Aggregate effects by retrieval status. Losses are large when state reaches control and small when it is maintained but never retrieved; diamonds denote access comparisons with no record to retrieve.

excluding it, our method averages 76.0% versus 73.0% for FrameSamp on the remaining Counting tasks.

The episode-level comparison in Figure 2 shows the same pattern: of the 115 Permanence episodes solved by exactly one system, ours solves 112, and of the 92 such Reference episodes, ours solves 86. Overall, the strongest gains arise when agentic tool use can accurately capture and retain task-relevant historical state that is no longer available from the current observation or a single retained frame. These system-level results do not isolate the effect of individual memory components. We therefore ablate state formation, updates, and retrieval in Figure 3 and Table 8.

4.3 RQ2: Which Components of State Maintenance Matter?

Different tasks depend on different maintained state. Matched ablations show strong task specificity (Figure 3). Removing route state reduces success by 68.0 points on PatternLock and RouteStick, while progress-count removal reduces SwingXtimes by 62.0 points. Removing demonstration-reference state lowers its three corresponding tasks by 44.7 points on average, and live bindings, demonstration bindings, and relation updates show the same localized pattern. In contrast, effects on unrelated tasks are small. Thus, different long-horizon tasks depend on different task-relevant state variables rather than a single generic memory representation. Exact task sets and aggregate effects are reported in Appendix Table 8.

State matters when it is used for control. The same mechanisms often remain active on tasks that never retrieve their state. On these comparisons, removing route state changes success by only -0.7 points, progress counting by 0.0, live binding by $+2.0$, and demonstration-reference or demonstration-binding state by 0.0 (Figure 3b). A separate 450-episode ablation gives the same pattern for perception: removing the identity feature anchor and optical-flow continuity check reduces success by 49.0 points when the maintained bindings are later retrieved, but changes performance by only $+1.5$ points on the tasks where the same tools run without retrieval (Figure 3b; per-task values in Appendix Table 11). Thus, forming or updating state alone is insufficient; its effect appears when that state is retrieved to resolve a control decision.

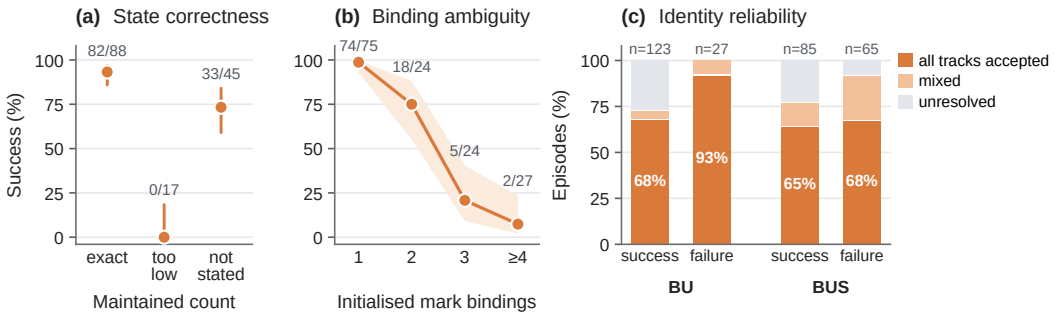


Figure 4: **Representative failure cases.** (a) SwingXtimes success by count accuracy. (b) PickHighlight success as initialized bindings increase. (c) Final carrier status for ButtonUnmask (BU) and ButtonUnmaskSwap (BUS). A carrier is a cover track bound to a remembered cube: "all tracks accepted" means every carrier ends internally verified or reacquired; "mixed" combines accepted and uncertain carriers; and "unresolved" means all carriers remain uncertain. These labels are internal states, not ground-truth identity checks. All panels pool three policy seeds and are diagnostic; bars in (a) and (b) are 95% Wilson intervals on the plotted rate.

Selective retrieval is a separate design problem. Replacing structured retrieval with a VLM decision at every proposed subgoal reduces overall success by 11.9 points: -17.2 points on tasks with retrievable historical state versus -3.0 points on the remaining tasks. Free-form retrieval can expose irrelevant historical state and overwrite subgoals that should instead be resolved from the current observation. Maintaining useful state and deciding when it should influence control are therefore distinct problems.

Together, the ablations support two conclusions: different tasks require different maintained state, and that state improves control only when it is selectively retrieved for a decision that depends on it. Additional robustness analyses are reported in Appendix D.2.

4.4 RQ3: What Challenges Remain Beyond Memory?

Figure 4 examines three representative failure cases in which memory interacts with state estimation and control. In (a), 82/88 SwingXtimes episodes with an exact count succeed, versus 0/17 when the count is too low; removing progress state also lowers success by 62 points. In (b), PickHighlight success falls from 74/75 episodes with one initialized binding to 2/27 with four or more, showing the difficulty of resolving identities and binding ambiguities. In (c), every carrier track ends internally accepted (verified or reacquired) in 25/27 ButtonUnmask and 44/65 ButtonUnmaskSwap failures, so an internally coherent track may still contain an incorrect binding or precede a control error. Other tasks couple memory with additional capabilities: StopCube benefits from dense local visual history (Recent4/16/32: 17.3/40.0/41.3% over three seeds; Appendix B), while MoveCube and InsertPeg require fine-grained imitation and execution. RouteStick, by contrast, reaches 53.3%, near its 55.6% oracle-subgoal ceiling. RoboMME therefore evaluates memory jointly with temporal perception, imitation, and control.

5 Conclusion and Limitations

We formulate robot memory as task-conditioned maintenance of policy-sufficient state and introduce SimpleARM, a training-free layer that specifies candidate state variables, maintains them online, retrieves them through structured access, and re-grounds recalled entities while leaving the subgoal predictor and VLA frozen. SimpleARM achieves 67.17% success on RoboMME, while matched ablations show that relation, reference, progress, and route state matter primarily where they are retrieved for control. These results support representing interaction-derived state rather than relying on retained visual history alone. SimpleARM targets episode-level state maintenance for hierarchical manipulation rather than high-frequency motion estimation or low-level control. Extending the schema to additional policy-relevant variables and evaluating it on real robots are important next steps.

AI USE STATEMENT

AI tools assisted with literature discovery and summarization, research ideation, experimental design and result interpretation, code implementation, and drafting and polishing. The authors reviewed and verified all AI-assisted code, calculations, claims, citations, and text and take responsibility for the final content.

ETHICS STATEMENT

All experiments use simulated robot environments; no human subjects or personal data are involved. The method has not been validated for safety-critical real-world deployment.

REPRODUCIBILITY STATEMENT

The appendix reports implementation details, configurations, seeds, task-level results, ablations, and the evaluation protocol.

REFERENCES

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025. URL https://arxiv.org/abs/2511.21631.

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 *Proceedings of the 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40, 2025a.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael R. Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. In *Proceedings of Robotics: Science and Systems*, 2025b. doi: 10.15607/RSS.2025.XXI.010. URL https://www.roboticsproceedings.org/rss21/p010.html.

Sage E. P. Boettcher, Daniela Gresch, Anna C. Nobre, and Freek van Ede. Output planning at the input stage in visual working memory. *Science Advances*, 7(13):eabe8212, 2021. doi: 10.1126/sciadv.abe8212. URL https://pubmed.ncbi.nlm.nih.gov/33762341/.

Jean-Yves Bouguet. Pyramidal implementation of the Lucas Kanade feature tracker: Description of the algorithm. Technical report, Intel Corporation, Microprocessor Research Labs, 2001.

Anthony Brohan et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of the 7th Conference on Robot Learning*, 2023. URL https://arxiv.org/abs/2307.15818.

Joya Chen, Zhaoyang Lv, Shiwei Wu, Kevin Qinghong Lin, Chenan Song, Difei Gao, Jia-Wei Liu, Ziteng Gao, Dongxing Mao, and Mike Zheng Shou. VideoLLM-online: Online video large language model for streaming video. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Chen_VideoLLM-online_Online_Video_Large_Language_Model_for_Streaming_Video_CVPR_2024_paper.html.

Tianxing Chen, Yuran Wang, Mingleyang Li, Yan Qin, Hao Shi, Zixuan Li, Yifan Hu, Yingsheng Zhang, Kaixuan Wang, Yue Chen, Hongcheng Wang, Junjie Wang, Tianhang Yang, Renjing Xu, Ruihai Wu, Yao Mu, Yaodong Yang, Hao Dong, and Ping Luo. RMBench: Memory-dependent robotic manipulation benchmark with insights into policy design. *arXiv preprint arXiv:2603.01229*, 2026. URL https://arxiv.org/abs/2603.01229.

Egor Cherepanov, Nikita Kachaev, Alexey K. Kovalev, and Aleksandr I. Panov. Memory, benchmark & robots: A benchmark for solving complex tasks with reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 9759–9815, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/1088fffae5f2aa8c8324e4f45f62248d-Abstract-Conference.html.

Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 183–188. AAAI Press, 1992.

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 *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2603.04639.

Anxhelo Diko, Tinghuai Wang, Wassim Swaileh, Shiyan Sun, and Ioannis Patras. ReWind: Understanding long videos with instructed learnable memory. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. URL https://arxiv.org/abs/2411.15556.

Yue Fan, Xiaojian Ma, Rujie Wu, Yuntao Du, Jiaqi Li, Zhi Gao, and Qing Li. VideoAgent: A memory-augmented multimodal agent for video understanding. In *European Conference on Computer Vision*, 2024. URL https://arxiv.org/abs/2403.11481.

Yue Fan, Xiaojian Ma, Rongpeng Su, Jun Guo, Rujie Wu, Xi Chen, and Qing Li. Embodied VideoAgent: Persistent memory from egocentric videos and embodied sensors enables dynamic scene understanding. In *IEEE/CVF International Conference on Computer Vision*, 2025. URL https://openaccess.thecvf.com/content/ICCV2025/html/Fan_Embodied_VideoAgent_Persistent_Memory_from_Egocentric_Videos_and_Embodied_Sensors_ICCV_2025_paper.html.

Xinying Guo, Chenxi Jiang, Hyun Bin Kim, Yuhang Han, Ying Sun, Yang Xiao, and Jianfei Yang. Chameleon: Control-indexed prospective memory for visuomotor manipulation. *arXiv preprint arXiv:2603.24576*, 2026. URL https://arxiv.org/abs/2603.24576.

Stephenie A. Harrison and Frank Tong. Decoding reveals the contents of visual working memory in early visual areas. *Nature*, 458:632–635, 2009. doi: 10.1038/nature07832. URL https://www.nature.com/articles/nature07832.

Zhenpeng Huang, Xinhao Li, Jiaqi Li, Jing Wang, Xiangyu Zeng, Cheng Liang, Tao Wu, Xi Chen, Liang Li, and Limin Wang. Online video understanding: OVBench and VideoChat-Online. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. URL https://openaccess.thecvf.com/content/CVPR2025/html/Huang_Online_Video_Understanding_OVBench_and_VideoChat-Online_CVPR_2025_paper.html.

Hongxiang Jiang, Zengrui Ge, Guo Chen, Qixiong Wang, Jile Jiao, Xuetao Feng, Yuan Wang, and Yan Wang. Memento: Toward an all-day proactive assistant for ultra-long streaming video. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/3b5f4587a0bdb81ecc6ce9d82320a5c2-Abstract-Conference.html.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998. doi: 10.1016/S0004-3702(98)00023-X.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713, 2025. URL https://proceedings.mlr.press/v270/kim25c.html.

Christopher A. Kurby and Jeffrey M. Zacks. Segmentation in the perception and memory of events. *Trends in Cognitive Sciences*, 12(2):72–79, 2008. doi: 10.1016/j.tics.2007.11.004. URL https://pubmed.ncbi.nlm.nih.gov/18178125/.

Yuna Kwak and Clayton E. Curtis. Unveiling the abstract format of mnemonic representations. *Neuron*, 110(11):1822–1828.e5, 2022. doi: 10.1016/j.neuron.2022.03.016. URL https://pubmed.ncbi.nlm.nih.gov/35395195/.

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*, 2026a. URL https://arxiv.org/abs/2605.10921.

Yuheng Lei, Zhixuan Liang, Hongyuan Zhang, and Ping Luo. VPWEM: Non-Markovian visuomotor policy with working and episodic memory. *IEEE Robotics and Automation Letters*, 11 (10):11283–11290, 2026b. doi: 10.1109/LRA.2026.3723315. URL https://ieeexplore.ieee.org/document/11652892.

Zhijia Liang, Jiaming Li, Weikai Chen, Yanhao Zhang, Haonan Lu, and Guanbin Li. OASIS: On-demand hierarchical event memory for streaming video reasoning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Liang_OASIS_On-Demand_Hierarchical_Event_Memory_for_Streaming_Video_Reasoning_CVPR_2026_paper.html.

Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, and Lei Zhang. Grounding DINO: Marrying DINO with grounded pre-training for open-set object detection. In *European Conference on Computer Vision*, 2024. URL https://arxiv.org/abs/2303.05499.

Lin Long, Yichen He, Wentao Ye, Yiyuan Pan, Yuan Lin, Hang Li, Junbo Zhao, and Wei Li. Seeing, listening, remembering, and reasoning: A multimodal agent with long-term memory. In *International Conference on Learning Representations*, volume 2026, pp. 146197–146246, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/ec370e882e8a02be955679ddbd2731b8-Abstract-Conference.html.

Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pp. 674–679, 1981.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024. URL https://arxiv.org/abs/2304.07193.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. SAM 2: Segment anything in images and videos. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2408.00714.

Yujiao Shen, Shulin Tian, Jingkang Yang, and Ziwei Liu. A simple baseline for streaming video understanding. *arXiv preprint arXiv:2604.02317*, 2026. URL https://arxiv.org/abs/2604.02317.

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*, 2026. URL https://arxiv.org/abs/2508.19236.

Lucy Xiaoyang Shi, Brian Ichter, Michael Robert Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, Adrian Li-Bell, Danny Driess, Lachy Groom, Sergey Levine, and Chelsea Finn. Hi Robot: Open-ended instruction following with hierarchical vision-language-action models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 54919–54933, 2025.

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*, 2026. URL https://arxiv.org/abs/2510.20328.

Marcel Torne, Karl Pertsch, Homer Walke, Kyle Vedder, Suraj Nair, Brian Ichter, Allen Z. Ren, Haohuan Wang, Jiaming Tang, Kyle Stachowicz, Karan Dhabalia, Michael Equi, Quan Vuong, Jost Tobias Springenberg, Sergey Levine, Chelsea Finn, and Danny Driess. MEM: Multi-scale embodied memory for vision language action models. *arXiv preprint arXiv:2603.03596*, 2026. URL https://arxiv.org/abs/2603.03596.

Edward K. Vogel, Andrew W. McCollough, and Maro G. Machizawa. Neural measures reveal individual differences in controlling access to working memory. *Nature*, 438:500–503, 2005. doi: 10.1038/nature04171. URL https://www.nature.com/articles/nature04171.

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*, 2023. URL https://arxiv.org/abs/2305.16291.

Xiaohan Wang, Yuhui Zhang, Orr Zohar, and Serena Yeung-Levy. VideoAgent: Long-form video understanding with large language model as agent. In *European Conference on Computer Vision*, 2024. URL https://arxiv.org/abs/2403.10517.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2502.12110.

Woongyeong Yeo, Kangsan Kim, Jaehong Yoon, and Sung Ju Hwang. WorldMM: Dynamic multimodal memory agent for long video reasoning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026. URL https://arxiv.org/abs/2512.02425.

Yufei Yin, Qianke Meng, Minghao Chen, Jiajun Ding, Zhenwei Shao, and Zhou Yu. VideoARM: Agentic reasoning over hierarchical memory for long-form video understanding. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026. URL https://arxiv.org/abs/2512.12360.

Haoji Zhang, Yiqin Wang, Yansong Tang, Yong Liu, Jiashi Feng, and Xiaojie Jin. Flash-VStream: Efficient real-time understanding for long video streams. In *IEEE/CVF International Conference on Computer Vision*, 2025. URL https://openaccess.thecvf.com/content/ICCV2025/html/Zhang_Flash-VStream_Efficient_Real-Time_Understanding_for_Long_Video_Streams_ICCV_2025_paper.html.

Yiyang Zhou, Yangfan He, Yaofeng Su, Siwei Han, Joel Jang, Gedas Bertasius, Mohit Bansal, and Huaxiu Yao. ReAgent-V: A reward-driven multi-agent framework for video understanding. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/de72a54c3d39523b0dd06b64ffa6d50d-Abstract-Conference.html.

A Additional Method and Implementation Details

A.1 Instruction-conditioned specification and access

The RoboMME implementation queries the instruction with the same frozen Qwen3-VL-4B-Instruct weights that serve as the subgoal predictor, with the grounded-subgoal adapter disabled and no image in the prompt, so the memory layer loads no additional model. Four queries run at the start of each episode. Two produce the state specification P: a prompted vote identifying candidate evidence sources and historical variables that may need to be maintained, and a single query reading the repetition structure of the instruction, whose count the progress gate later uses. Two more produce the retrieval specification A: one naming how each monitored entity is distinguished, and one emitting zero or more schema-validated operations for marked entities, previously handled objects, event-relative progress, or ordered routes. Each of the three voted stages issues the same prompt at three line widths and takes the majority; identical wording under different line breaking is what separates stable answers from unstable ones, since decoding is deterministic. A specification is adopted only when at least two of the three readings call for it, and the reader program additionally requires two schema-valid readings: the reader compiler returned three valid readings in 1,614 of the 2,400 final episodes, two in 552, and fell back to no program on the remaining 234. Language therefore determines what state may be useful and how it may later be queried, but does not assign stored values or force memory retrieval. Perceptual and event evidence establish state online, while the structural query in Sec. 3.3 determines whether maintained history is used for the current subgoal. Demonstrations, when present, are handled by the corresponding perception and procedure parsers rather than by these instruction stages.

Compilation and policy-facing access are separate. A compiled operation is applied only when a structural test identifies a compatible field in the proposed subgoal. In the final evaluation, reader programs are applied on PickHighlight, VideoRepick, and VideoPlaceButton; programs emitted on several other tasks remain inert, and PickXtimes falls back to no program. Entity records are not injected merely because they exist, while counts, ordinals, and routes use their corresponding field parsers. This implements the general contract of Eq. 11: read memory when a proposed action field is underdetermined by the current view and determined by interaction history. The concrete implementation still contains RoboMME-specific lexical choices—token tests on the composer's phrase, regular expressions for route and ordinal slots, an appearance vocabulary, and event keywords in demonstration parsers. These instantiate the access interface; they are not the framework's definition of robot memory.

No language model participates in the remaining operators: the update $\mathcal{U}$, the retrieval rule $\mathcal{R}$, and the verification and re-grounding step $\mathcal{V}$ are perception tools, event predicates, and the structural field test, so every language-model call in the method occurs before the first action is taken.

The plan's coarse NEED field is affirmative in all 2,400 final episodes and therefore does not gate writes in this benchmark. Its WATCH phrases and the observable pixel or event predicates provide the discriminating decisions. This is why we describe the VLM as proposing what to monitor and the tools as verifying the resulting state.

A.2 Typed schemas, verification, and state lifecycle

Table 2 expands the three stores in Eq. 7. All records are episode-scoped. A candidate write is admitted only after its type-specific observable predicate is satisfied. Entity records then move through *candidate*, *verified*, *uncertain*, and *reacquired* states. Historical identity can therefore persist while current geometry is withheld. A verified read supplies historical disambiguation; entity reads additionally require current-view localization before modifying the subgoal. If a compatible field, verified state, or safe current grounding is absent, the memory abstains.

State semantics. The three stores differ in the temporal structure they preserve. Entity and relation state retains historical identity and task-relevant relations—for example, an object's role, container association, or demonstrated identity—rather than the observation in which that relation was established. Current geometry may become uncertain while the historical identity or relation remains valid; when needed for control, the corresponding referent is re-grounded in the current view.

Table 2: Typed memory schemas in the RoboMME instantiation. Language selects fields to monitor; frozen tools supply and verify their values.

state type	persistent field	verified update	policy-facing read
entity / relation	identity, role, container or current relation	detector proposal plus appearance/geometric consistency	historical referent; current geometry is re-grounded
event / progress	event count, repetition index or event-relative stage	task-observable event predicate	count, ordinal or completion field
trajectory / procedure	ordered route elements and progress index	parsed demonstration segment or verified route transition	next waypoint or ordered procedural element

Progress state represents quantities induced by interaction history rather than by any single observation. For example, a counter can be updated on each newly verified event occurrence as

$$c_t = c_{t-1} + \mathbf{1}[e_t \text{ is newly verified}]\,. \tag{13}$$

Counts, repetition indices, and event-relative stages therefore summarize verified transitions accumulated over the episode.

Trajectory and procedural state preserves order-sensitive structure. We represent an ordered procedure as

$$T = (\tau_1, \ldots, \tau_m), \qquad j_t \in \{1, \ldots, m\}, \tag{14}$$

where τ_i denotes a waypoint, route segment, or procedural element and j_t records current progress when applicable. Order is itself part of the maintained state: the same observation or waypoint may imply different subsequent behavior at different positions in the procedure.

A.3 RoboMME tool instantiation

Entity and relation state. Grounding DINO (Liu et al., 2024) proposes candidate boxes, and a calibrated named-colour pixel predicate verifies candidates before admission. DINOv2-small features (Oquab et al., 2024) provide an independent identity anchor. Updates combine identity and geometric continuity; pyramidal Lucas–Kanade optical flow (Lucas & Kanade, 1981; Bouguet, 2001) is consulted when motion continuity is needed, while SAM2.1-tiny masks (Ravi et al., 2025) support ambiguous association and recovery. A recovery can widen the search at most three times. Agreement restores a record to *verified*; disagreement preserves historical identity while marking current geometry *uncertain*. When a retrieved identity is needed for control, current-view detection and localization provide executable geometry; if reliable grounding cannot be recovered, memory abstains. Motion and mask tools are invoked only when required by the update or recovery process.

Progress and trajectory state. RGB event predicates update repetition and flash counts. Demonstration parsers record handled objects and event-relative targets. The route parser maintains an ordered sequence of spatial segments together with its current progress index. This representation contains neither demonstrated grasp style nor insertion side; tasks requiring these additional capabilities are discussed in RQ3.

Fixed settings. The implementation uses write threshold 0.30, DINO similarity threshold 0.65, recovery budget 3, maximum episode length 1,300, and action horizon 16. The policy is GroundSG checkpoint 79,999 under configuration `mme_vla_suite`; the composer is Qwen3-VL-4B-Instruct with grounded-subgoal adapter checkpoint 1,200. All models, prompts, thresholds, and tools remain fixed during evaluation, and M_0 is reset for every episode.

The added mechanism issues exactly ten language-model calls per episode, in every one of the 2,400 final episodes: three for the state specification, three for entity distinguishability, three for the reader program, and one for the repetition count. All ten are text-only prompts to the frozen composer weights with the adapter disabled, and all ten precede the first action, so the method adds no per-step language-model cost. Perception tools use 2.02 seconds per episode on average: 1.10 seconds for detection, 0.65 for visual features, 0.25 for masks, and 0.03 for optical flow. These costs exclude the frozen composer and VLA shared with the base system.

Table 3: **Recent-frame baselines over three policy seeds.** RoboMME's FrameSamp+ModuL architecture trained and evaluated with the latest 4, 16, or 32 native frames. Each variant is trained once (seed 42) and its final checkpoint is evaluated with policy seeds 7, 11, and 23, 16 tasks × 50 episodes per seed. Task cells are three-seed mean success (%); the lower block gives each seed's successes out of 800 and the mean and sample standard deviation across seeds. The Recent32 column is the one reported in Table 1.

task	Recent4	Recent16	Recent32
Counting			
BinFill	31.3	36.7	34.0
PickXtimes	74.0	80.0	83.3
SwingXtimes	83.3	78.0	80.7
StopCube	17.3	40.0	41.3
Permanence			
VideoUnmask	28.0	29.3	19.3
ButtonUnmask	26.7	28.7	20.0
VideoUnmaskSwap	22.0	28.0	23.3
ButtonUnmaskSwap	16.0	9.3	28.0
Reference			
PickHighlight	18.7	28.7	26.7
VideoRepick	14.0	18.7	26.7
VideoPlaceButton	32.0	30.0	26.0
VideoPlaceOrder	25.3	23.3	24.7
Imitation			
MoveCube	35.3	42.0	52.0
InsertPeg	0.7	0.7	2.0
PatternLock	8.7	11.3	8.7
RouteStick	6.7	6.7	7.3
seed 7 (of 800)	222	243	254
seed 11 (of 800)	230	250	254
seed 23 (of 800)	208	244	248
mean ± sd (%)	$\mathbf{27.50} \pm 1.39$	$\mathbf{30.71} \pm 0.47$	$\mathbf{31.50} \pm 0.43$

B Evaluation Protocol

The final benchmark comprises three policy seeds and 50 episodes per task and seed. Table 4 reports every seed-level result together with the standard deviation across the three seeds; the sixteen-task mean is 67.17% with a seed-to-seed standard deviation of 1.23 points. Published RoboMME rows are compared at the task and suite aggregate. Separately, we evaluate the released FrameSamp-Modul checkpoint at seed 7 on the same 800 episode identities as our method. The two runs share task, scene, and demonstration but were executed on different hosts, so the paired comparison controls episode difficulty without being a same-session replication.

Recent$N$+ModuL is RoboMME's FrameSamp+ModuL architecture trained and evaluated with the causal suffix of the latest N native observations, using 16 history tokens per frame (64, 256, and 512 active history tokens for $N = 4$, 16, and 32). Each variant is trained once from the $\pi_{0.5}$ base with training seed 42 for 80,000 updates, and its checkpoint 79,999 is evaluated with the same three policy seeds as our method (7, 11, and 23; 16 tasks × 50 episodes per seed). Its 2,400 episodes therefore match our evaluation protocol, but the three rounds repeat one trained model rather than three training runs. Table 3 lists every round: Recent32 succeeds on 254, 254, and 248 of 800 episodes ($31.50 \pm 0.43\%$), Recent16 on 243, 250, and 244 ($30.71 \pm 0.47\%$), and Recent4 on 222, 230, and 208 ($27.50 \pm 1.39\%$). The seed-7 rounds and the seed-11/23 rounds ran on different hosts, so host effects are not separated from seed effects. On StopCube, Recent4, Recent16, and Recent32 succeed on 26, 60, and 62 of 150 episodes (17.3, 40.0, and 41.3%); these values are used only as a descriptive diagnostic of recent temporal context.

Each of the ten final-method ablations uses seed 11, covers all sixteen tasks, and matches 50 ablated episodes per task to full-method controls with identical task and episode identifiers. We report the ablated-minus-control success difference together with the number of matched pairs whose outcome

Table 4: Per-seed success (%) for the final method. Each seed contains 50 episodes for each of the sixteen tasks; the last column reports the sample standard deviation across seeds.

task	seed 7	seed 11	seed 23	mean	sd
BinFill	72	54	54	60.00	10.39
PickXtimes	88	92	94	91.33	3.06
SwingXtimes	78	76	76	76.67	1.15
StopCube	0	0	0	0.00	0.00
VideoUnmask	100	94	94	96.00	3.46
ButtonUnmask	76	90	80	82.00	7.21
VideoUnmaskSwap	86	78	76	80.00	5.29
ButtonUnmaskSwap	52	60	58	56.67	4.16
PickHighlight	68	70	60	66.00	5.29
VideoRepick	74	84	76	78.00	5.29
VideoPlaceButton	82	82	82	82.00	0.00
VideoPlaceOrder	88	88	88	88.00	0.00
MoveCube	66	70	66	67.33	2.31
InsertPeg	2	6	4	4.00	2.00
PatternLock	92	96	92	93.33	2.31
RouteStick	60	48	52	53.33	6.11
AVG	**67.75**	**68.00**	**65.75**	**67.17**	**1.23**

changes in each direction. Predicted-active sets are fixed from the code path and control-run activity before aggregation; all remaining tasks are retained as negative controls. Each ablation is run at a single policy seed, so an effect is characterized by the size of the matched difference and by whether it is concentrated on the predicted-active tasks, not by a multi-seed replicate. Where a figure draws an interval, it is a 95% interval on the plotted quantity—a percentile interval from 20,000 paired bootstrap resamples for a matched difference, and a Wilson interval for a single success rate—summarizing the spread of the evaluated episode set rather than serving as a test.

Structured diagnostics record state transitions, reads, abstentions, recovery events, tool activity, and episode outcomes. They do not contain the source images or a complete step-aligned oracle action trace. Consequently, they support analyses of internal state and associations with outcome, but cannot visually certify every accepted identity or uniquely attribute every residual failure to grounding versus downstream control.

Two references calibrate how large a pooled difference has to be before it is worth interpreting. One behaviorally equivalent evaluation pair preserves reader routing on all 2,400 episodes yet flips 11.9% of individual outcomes, changing the mean by only -0.42 points; and cells of the final ablations that are inactive for a task range from roughly -3.4 to $+0.9$ points. Both are comparable to the 1.23-point seed-to-seed standard deviation of the final method. We therefore read pooled effects of this scale as run-to-run variability unless a localized mechanism and a matched task signature support them.

C Extended Benchmark Results

C.1 Task-state annotation

Table 5 records the smallest history-dependent variables implied by each task's instruction and oracle subgoal template. The labels are defined from task semantics rather than observed performance, and they may overlap. For example, ButtonUnmaskSwap combines a persistent relation with progress, while VideoPlaceButton combines historical identity with event-relative order. MoveCube and InsertPeg require manipulation choices not encoded by our route state.

The annotation explains where the system comparison is expected to favor explicit state, but it is not itself causal evidence. In particular, VideoUnmask and BinFill show large or moderate gains over FrameSamp despite weak sensitivity to the tested memory ablations; PickHighlight's mark pathway has no removal ablation. The all-task ablations below supply the causal attribution.

Table 5: Task-state annotation derived from the oracle subgoal templates. Labels may overlap. * marks state expressed only by the instruction, not by the oracle subgoal text.

suite	task	history-dependent state
Counting	BinFill	count/progress
	PickXtimes	count/progress
	SwingXtimes	count/progress
	StopCube*	count/progress
Permanence	VideoUnmask	entity–location relation
	ButtonUnmask	entity–location relation
	VideoUnmaskSwap	entity–location relation after swap
	ButtonUnmaskSwap	entity–location relation; count/progress
Reference	PickHighlight	marked entity; count/progress
	VideoRepick	demonstrated entity identity; count/progress
	VideoPlaceButton	demonstrated entity identity and event-relative order
	VideoPlaceOrder	demonstrated entity identity and ordinal order
Imitation	MoveCube	demonstrated manipulation procedure
	InsertPeg	demonstrated grasp/insertion procedure
	PatternLock	ordered spatial route
	RouteStick	ordered spatial route and turn direction

Table 6: **Episode-matched comparison with the released FrameSamp-Modul checkpoint.** The runs share task, scene, demonstration, and episode identity but were executed on different hosts. Tasks are grouped into the four RoboMME suites; both halves of the table have the same columns. *FS* abbreviates FrameSamp-Modul, Δ is ours minus FrameSamp in percentage points, and *only* counts the episodes solved by exactly one system, as ours-only / FrameSamp-only.

task	ours	FS	Δ	only	task	ours	FS	Δ	only
Counting					*Reference*				
BinFill	36/50	22/50	+28	17/3	PickHighlight	34/50	9/50	+50	26/1
PickXtimes	44/50	48/50	−8	1/5	VideoRepick	37/50	17/50	+40	22/2
SwingXtimes	39/50	44/50	−10	4/9	VideoPlaceButton	41/50	28/50	+26	15/2
StopCube	0/50	25/50	−50	0/25	VideoPlaceOrder	44/50	22/50	+44	23/1
suite	119/200	139/200	−10.0	22/42	*suite*	156/200	76/200	**+40.0**	86/6
Permanence					*Imitation*				
VideoUnmask	50/50	17/50	+66	33/0	MoveCube	33/50	41/50	−16	6/14
ButtonUnmask	38/50	10/50	+56	28/0	InsertPeg	1/50	5/50	−8	1/5
VideoUnmaskSwap	43/50	13/50	+60	32/2	PatternLock	46/50	26/50	+40	20/0
ButtonUnmaskSwap	26/50	8/50	+36	19/1	RouteStick	30/50	34/50	−8	7/11
suite	157/200	48/200	**+54.5**	112/3	*suite*	110/200	106/200	+2.0	34/30
All 16 tasks:	ours **542/800**	FS **369/800**	$\Delta = +\mathbf{21.6}$	only **254/81**					

C.2 Episode-matched FrameSamp comparison

Table 6 lists every task. The reproduced checkpoint reaches 46.12%, close to the published three-seed 44.51%. The paired difference is 21.6 points, with 254 ours-only and 81 FrameSamp-only successes. Ten of the sixteen tasks favor our method by at least 26 points; StopCube is the only large loss. The suite rows locate the gain: +54.5 points on Permanence and +40.0 on Reference, against +2.0 on Imitation and −10.0 on Counting. Because the runs share episode identities but not host-level execution noise, this is a paired comparison of the two systems at the task level, not a same-session replication.

D Full Mechanism Ablations

All ten ablations in Table 7 cover the same sixteen tasks and 800 matched episodes. Figure 5 shows the pooled effects; Table 9 gives every task cell. The pooled score is useful for overall impact, but the active/off-target split in Table 8 is the direct test of mechanism specificity. Table 8 reports the task sets and aggregate matched effects used in the RQ2 analysis. Task sets are fixed from the state

Table 7: All final-method ablations on the common 800-episode seed-11 pool. Each row compares the ablation with its episode-matched full-method control, ordered by the size of the effect. *better/worse* counts the matched pairs whose outcome changes in each direction, so their sum is the number of episodes the ablation altered at all. Row order matches Figure 5.

ablation	control	ablated	Δ	better/worse
Memory components				
agent-decided access	67.38	55.50	**−11.88**	28/123
no route state	67.75	56.25	**−11.50**	24/116
no demo references	67.25	58.12	**−9.12**	43/116
no live identity track	67.50	61.62	−5.88	42/89
no demo notes	67.75	62.38	−5.38	38/81
no flash-count state	67.62	62.62	−5.00	40/80
no demo following	67.75	64.88	−2.88	42/65
Robustness and implementation controls				
single-vote write plan	67.62	64.62	−3.00	39/63
refine every verb	67.38	65.38	−2.00	47/63
source from environment	67.25	67.25	0.00	44/44

Table 8: **Task specificity of memory-component ablations.** Each ablation is evaluated on all 16 tasks with 50 matched episodes per task. Task sets are specified *a priori* from the state variable or access mechanism required by the final method. Δ denotes the change in success rate relative to the full method, in percentage points.

Category	**Ablation**	**Tasks requiring component**	**Δ required**	**Δ remaining**
Maintained state	Route-state removal	PatternLock, RouteStick	**−68.0**	−3.4
	Progress-count removal	SwingXtimes	**−62.0**	−1.2
	Demonstration-reference removal	VideoRepick, VideoPlaceButton, VideoPlaceOrder	**−44.7**	−0.9
	Live-binding removal	ButtonUnmask, ButtonUnmaskSwap	**−53.0**	+0.9
	Demonstration-binding removal	VideoUnmask, VideoUnmaskSwap	**−28.0**	−2.1
State update	Demonstration relation update	VideoUnmaskSwap	**−50.0**	+0.3
Memory access	Agent-decided access	Ten note-bearing tasks	**−17.2**	−3.0

variable or access mechanism exercised by the final method; all remaining tasks serve as negative controls.

The matrix reveals both necessity and negative-control specificity. PatternLock and RouteStick lose 90 and 46 points without route state; SwingXtimes loses 62 without the flash count; the three demonstration-reference tasks lose 34–52 without reference state; the ButtonUnmask pair loses 64 and 42 without the live binding; and VideoUnmaskSwap loses 50 without demonstration cover-following. Off-target changes are much smaller. The largest of them—no route on ButtonUnmaskSwap (−16 points)—occurs on a task where route state is never formed, so it cannot be an effect of removing that state; single cells of this size are read against the run-to-run variability band in Appendix B rather than on their own.

D.1 Activity, access, and compiled readers

Table 10 lists how often each mechanism runs. Live carrier bindings are formed in 44–77% of Counting episodes, and the flash counter observes red cues on route tasks, yet removing those mechanisms

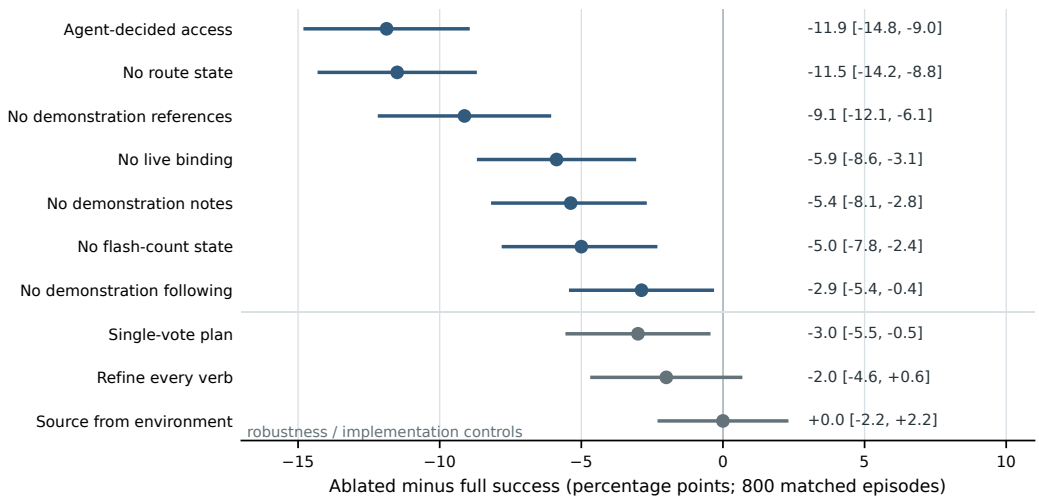


Figure 5: All ten final-method ablations on 800 episode-matched comparisons each. Points are ablated-minus-full success and lines are paired-bootstrap 95% intervals. The lower three rows are robustness controls rather than typed memory mechanisms.

Table 9: Complete ablation–task matrix. Every cell contains 50 matched episodes at seed 11; entries are ablated minus full success (percentage points). Bold cells are predicted active before observing outcomes. Task abbreviations follow their column order in Table 1.

	BF	PX	SX	SC	VU	BU	VUS	BUS	PH	VR	VPB	VPO	MC	IP	PL	RS
no route	+4	0	−12	0	+2	−12	+2	−16	0	−8	0	−2	0	−6	**−90**	**−46**
no flash count	+10	−6	**−62**	0	+4	−2	0	−6	−2	−10	0	0	0	−6	−2	+2
no demo references	+6	−6	0	0	0	0	0	0	+2	**−48**	**−34**	**−52**	−2	−6	−4	−2
no live track	0	+6	0	0	+2	**−64**	+2	**−42**	+2	0	0	−2	+2	−4	−2	+6
no demo notes	+2	+2	−4	0	**−8**	−4	**−48**	−10	+4	−8	0	0	−6	−6	0	0
no demo following	+2	+2	−2	0	−6	+2	**−50**	+2	−4	0	0	−2	+10	−4	0	+4
agent access	**−12**	**−6**	**−38**	0	**−2**	**−54**	**−8**	**−56**	−6	**+4**	**0**	**0**	−4	−4	−6	+2
single vote	−4	−2	−8	0	−4	−10	−8	−12	+2	−4	0	−2	+6	0	−2	0
refine all verbs	+6	−2	+4	0	0	−20	−4	−22	+4	−2	0	0	+6	−4	−8	+10
source = environment	+8	−2	−6	0	+2	−4	+2	−2	−6	0	0	−2	+8	−2	0	+4

has no corresponding effect there. Similarly, demonstration notes are written in 92–96% of VideoPlaceButton and VideoPlaceOrder episodes but are never read, and their removal changes neither task. Activity therefore confirms that a mechanism is exercised, but does not establish a contribution to control.

Structured access prevents active but irrelevant state from reaching control. Under the structural rule, entity reads per episode are 0.0 on BinFill, PickXtimes, and SwingXtimes; under agent-decided access they rise to 37.2, 35.5, and 29.0. On ButtonUnmask and ButtonUnmaskSwap they rise from 12.9 to 34.0 and from 16.9 to 43.0. In worsened SwingXtimes pairs, the free-form reader exposes a plausible carrier track to pick and move subgoals, replacing the current-view coordinate and disrupting the subsequent count. This explains why structured access is a mechanism rather than merely a cheaper retrieval policy.

The reader compiler is deliberately separated from this gate. Its programs are correctly applied on PickHighlight (mark), VideoRepick (handled object), and VideoPlaceButton (sequence). It also emits inert programs on seven live or non-reference tasks, falls back on all PickXtimes episodes, and falls back on half of BinFill. These outputs do not affect control unless the structural gate finds a compatible history-dependent field. The evidence therefore supports agentic candidate specification plus precision-oriented access, not a claim of perfect compiler routing.

Table 10: Mechanism activity does not imply contribution to control. Activity is the fraction of final-run episodes in which the mechanism is exercised; sensitive tasks are those affected by the corresponding ablation.

mechanism	active tasks (fraction of episodes)	ablation-sensitive tasks
live carrier track	BinFill 44%, PickXtimes 71%, SwingXtimes 77%, ButtonUnmask 100%, ButtonUnmaskSwap 100%	ButtonUnmask, ButtonUnmaskSwap
demo cover track	VideoUnmask 100%, VideoUnmaskSwap 100%	VideoUnmaskSwap
flash counter	SwingXtimes 96%, PatternLock 47%, RouteStick 47%	SwingXtimes
route fill	PatternLock 100%, RouteStick 100%	PatternLock, RouteStick
entity read	VideoUnmask 96%, ButtonUnmask 100%, VideoUnmaskSwap 92%, ButtonUnmaskSwap 100%, VideoRepick 100%	ButtonUnmask, ButtonUnmaskSwap
mark reader	PickHighlight 100%	not tested
SAM	seven tasks, 54–79%	not tested

Table 11: **Matched identity-tool ablation.** The frozen feature anchor and optical-flow continuity check are removed while the color verifier is retained. The comparison covers nine tasks with 50 matched episodes each. "Retrieved" means that the maintained binding is read at decision time and conditions the action; the pooled "maintained, not retrieved" row is the open circle of Figure 3b (the four tasks on which the tools run in at least 20% of episodes), and "all other tasks" adds VideoPlaceOrder, where they run in 8%. SAM is absent from both arms, so this comparison should not be interpreted as removing every identity tool.

Task	Binding retrieved	Full	Ablated	Δ
ButtonUnmask	yes	42/50	9/50	$\mathbf{-66}$
ButtonUnmaskSwap	yes	34/50	10/50	$\mathbf{-48}$
VideoUnmaskSwap	yes	37/50	14/50	$\mathbf{-46}$
VideoRepick	yes	34/50	16/50	$\mathbf{-36}$
VideoUnmask	no	48/50	48/50	0
SwingXtimes	no	34/50	33/50	-2
PickXtimes	no	46/50	48/50	$+4$
BinFill	no	26/50	28/50	$+4$
VideoPlaceOrder	no	42/50	43/50	$+2$
Retrieved tasks		147/200	49/200	$\mathbf{-49.0}$
Maintained, not retrieved (4 tasks)		154/200	157/200	$+1.5$
All other tasks (5 tasks)		196/250	200/250	$+1.6$

Table 11 reports the per-task values underlying the identity-tools comparison in Figure 3b. Unlike the typed-state removals, this ablation is exercised across both groups: it produces large losses when the maintained binding is retrieved and almost no change when the same tools run without the binding reaching control.

D.2 Robustness controls

Forcing one planning vote changes the pooled score by -3.0 points, refining every verb by -2.0, and forcing the evidence source to environment by 0.0. The first two pooled differences are comparable with the between-session drift band and are not treated as core mechanisms. The refinement control nevertheless has a localized timing effect: ButtonUnmask and ButtonUnmaskSwap lose 20 and 22 points, because refining button-press coordinates increases composer calls and delays the prerequisite before the historical container read. This is evidence about downstream execution, not about memory content.

Table 12 separates robustness analyses from the final component ablations. Instruction rewording produces the largest sensitivity; providing the evidence source from the environment reduces, but does not eliminate, that loss. Detector substitution, write-policy changes, removing the write gate, and loosening the structural access test have smaller effects. Because these comparisons use a distinct evaluation configuration, we do not combine them with the matched final-method ablations in Figure 3 or Table 8.

Table 12: **Supplementary robustness comparisons.** Each row is matched by task and episode to a configuration-matched control. These comparisons use a distinct evaluation configuration and therefore characterize sensitivity rather than final-method component necessity. Δ is variant minus control in success percentage points.

Comparison	Tasks	Matched episodes	Δ
Instruction rewording	8	416	-17.8
with environment-provided evidence source	8	416	-7.7
Detector substitution	8	165	-5.5
Keyword rather than agent-specified write policy	11	422	-4.5
Write gate removed	11	548	-5.1
Structural access replaced by loose lexical access	6	222	-5.9

Table 13: Matched seed-7 no-flow control on five identity/update tasks. Both arms ran on a host where SAM was unavailable, unlike the final method; therefore this does not estimate flow's marginal effect in the shipped tool stack. Δ is no-flow minus control, and *better/worse* counts the matched episodes whose outcome changes in each direction.

scope	update condition	control $\rightarrow$ no flow	Δ	better/worse
ButtonUnmaskSwap	bound cover moves after occlusion	$34/50 \rightarrow 22/50$	-24	2/14
other four tasks	no live post-cover update of this type	$161/200 \rightarrow 161/200$	0	12/12
pooled	five predefined tasks	$\mathbf{195/250 \rightarrow 183/250}$	$\mathbf{-4.8}$	14/26

D.3 Optical-flow control under a different tool stack

The no-flow comparison (Table 13) is not part of the final ten-ablation set. Both its ablated and control arms ran on a host where SAM could not be imported, whereas SAM is available in the final method and is used for ambiguous association and reacquisition. The localized ButtonUnmaskSwap loss is therefore evidence that flow can support motion continuity when no mask fallback is available; it does not establish that flow is necessary in the final evaluated configuration. A matched no-flow run with SAM available would be required for that claim.

E Failure Diagnostics and State Coverage

E.1 Representational coverage

Table 14 summarizes representative tasks for which the current schema does or does not contain the required control variable. It interprets task requirements together with the matched ablations and oracle headroom; it is not an additional performance comparison.

E.2 Memory-state attainment across tasks

Table 15 summarizes how far failed episodes engage the available memory states. The categories are computed from episode-level totals and therefore report the furthest attained state or access event, not when or why failure occurs. In 12 of 16 tasks, successful and failed episodes receive essentially the same attainment label; 559/788 failures occur on tasks for which that label is constant. The table is therefore a coverage description rather than a failure localization. VideoPlaceButton and VideoPlaceOrder use a specialized demonstration-reference path that is not represented by the common entity/event-access statistic, so their access is not assigned to a directly comparable category. Matched ablations in RQ2, rather than these observational counts, provide causal evidence.

E.3 State correctness and binding complexity

For SwingXtimes goals that specify two or three repetitions, all 17 episodes with an under-counted state fail, while 82/88 with the exact count succeed. Combined with the matched no-count ablation,

Table 14: **Representational coverage of the current memory schema.** Matched removals quantify the contribution of represented variables; uncovered tasks require a control variable absent from the schema. "Represented" does not imply that the complete controller succeeds.

task	required policy variable	represented?	evidence
SwingXtimes	accumulated swing count	yes	no count: -62 points
ButtonUnmask	live cube–container binding	yes	no binding: -64 points
VideoPlaceOrder	demonstrated ordinal target	yes	no reference: -52 points
PatternLock	ordered route	yes	no route: -90 points
RouteStick	ordered route and turn sense	yes	53.3%, near 55.6% oracle ceiling
StopCube	timed reach count of moving cube	no	0/150; no state formed
MoveCube	demonstrated manipulation manner	no	no tested ablation changes outcome materially
InsertPeg	demonstrated peg end and insertion side	no	no corresponding memory field

Table 15: Memory-state attainment among the 788 failed episodes. Each count records the furthest stage observed at any point in the episode, not the stage that caused failure. Specialized reference access in VideoPlaceButton and VideoPlaceOrder is not represented by the same access statistic and is therefore reported separately.

task	failures	furthest recorded attainment (episodes)	task-level interpretation
BinFill	60	state formed, no recorded access: 60	count state maintained
PickXtimes	13	state formed, no recorded access: 13	count state maintained
SwingXtimes	35	state formed; ordinal access analyzed separately	progress state maintained
StopCube	150	no usable typed state: 150	required timed state absent
VideoUnmask	6	state returned: 5; formed without access: 1	demonstration relation
ButtonUnmask	27	state returned: 27	live entity–carrier binding
VideoUnmaskSwap	30	state returned: 21; all reads abstained: 3; no access: 6	moving demonstration relation
ButtonUnmaskSwap	65	state returned: 65	live moving binding
PickHighlight	51	mark relation returned: 51	historical mark binding
VideoRepick	33	reference returned: 33	demonstrated identity
VideoPlaceButton	27	specialized reference path; common access statistic unavailable	demonstrated target
VideoPlaceOrder	18	specialized reference path; common access statistic unavailable	demonstrated ordinal target
MoveCube	49	no usable state: 40; formed without access: 9	demonstrated manner absent
InsertPeg	144	no usable state: 132; formed without access: 12	peg end/side absent
PatternLock	10	route returned: 10	ordered route
RouteStick	70	route returned: 70	ordered route

this links missed event detections to failure without claiming that an exact count is sufficient. For PickHighlight, initialized bindings of one, two, three, and at least four yield 74/75, 18/24, 5/24, and 2/27 successes. The trend is diagnostic because no matched ablation removes the mark-memory pathway.

On ButtonUnmask, failures average 7.19 rejected tracks and 17.48 exhausted recoveries per episode, versus 1.23 and 2.63 for successes. On ButtonUnmaskSwap the corresponding averages are 12.71 and 40.22 versus 2.92 and 4.24. Only 6% and 10% of failed-episode read attempts abstain, and every carrier ends internally accepted in 25/27 and 44/65 failures (verified in 22/27 and 39/65, the rest reacquired; Figure 4c). Internal verification reflects the method's identity and geometry checks, not ground-truth identity. The pattern is consistent with identity errors that remain undetected or with later control failure; the available diagnostics do not separate these possibilities.

E.4 Route length and downstream execution

Table 16 slices the two route tasks by parsed route length. PatternLock remains reliable through route length three and then becomes less stable. RouteStick declines broadly from lengths two through six, although the small length-seven cell is non-monotonic. Route length is not randomized and can correlate with geometry, so these are difficulty slices rather than causal length effects. None of the 10 PatternLock or 70 RouteStick failures reaches the 80-subgoal interaction horizon. By

Table 16: Final-method success by parsed route length. These are observational difficulty slices, not controlled ablations; small non-monotonic cells, especially length seven, should not be overread.

	PatternLock		RouteStick	
route length	n	success	n	success
1	24	100.0%	—	—
2	27	100.0%	42	76.2%
3	54	100.0%	36	58.3%
4	24	91.7%	27	40.7%
5	3	33.3%	27	25.9%
6	12	75.0%	9	11.1%
7	6	50.0%	9	88.9%

contrast, MoveCube reaches that horizon in 37/49 failures (75.5%) and InsertPeg in 61/144 (42.4%), indicating missing schema fields together with difficult downstream execution.

E.5 Seed-invariant demonstration failures

VideoRepick's internal `correct cube` record is marked verified in all 150 episodes, including all 33 failures. VideoPlaceButton has the same 41 successful episode identities at all three policy seeds; VideoPlaceOrder shares 43 of 44 successful identities. These outcomes suggest that residual demonstration-reference failures are fixed primarily by how the demonstration is interpreted, rather than by policy-seed variation. Internal verification still does not certify that the selected demonstration entity is correct.

E.6 Episode-level state-to-action lifecycle

Figure 6 follows a complete ButtonUnmaskSwap episode. Unlike the aggregate results, this example shows when state is written, updated, accessed, and re-grounded before it is passed to the VLA. It is qualitative evidence of mechanism operation, not an estimate of accuracy or necessity.

E.7 Tool-use examples

Figures 7–13 provide complementary successful examples for the binding, progress, reference, route, and mark mechanisms. Each combines rollout frames with a trace transcribed from the episode log. Red borders denote demonstration frames, the yellow dot is the coordinate sent to the VLA, and overlays mark memory coordinates (○), composer coordinates (×), entity–cover bindings (□), and event or cue locations (◇). Figures 7–11 are seed-7 replications of the final configuration on a host where the mask tool was unavailable; the identity and geometric checks sufficed in the illustrated reads. Figures 12–13 use the same route and mark readers as the final method but come from an earlier generic-gate run, before the unified entity tool stack, which those tasks do not invoke.

E.8 Remaining causal questions

The current ablations establish the necessity and specificity of several typed variables and the access rule. They do not isolate the PickHighlight mark reader, SAM, or current-view grounding, nor do they uniquely decompose each failed episode between correct state, correct access, grounding, and control. A mark-reader removal would test the one large unablated pathway, and a matched oracle ladder—correct state, then oracle access, then oracle grounding—would provide finer attribution. No missing cell is inferred from invocation frequency or internal verification.

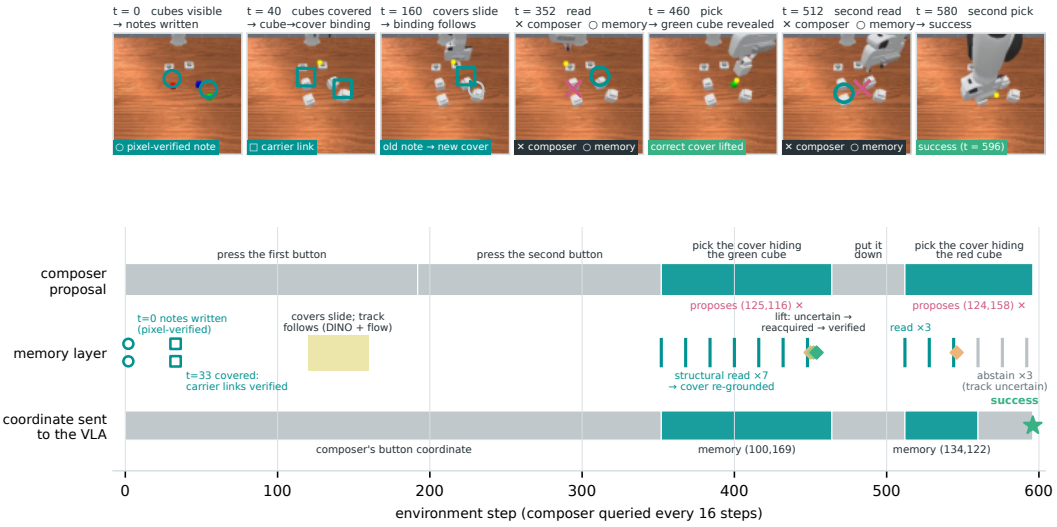
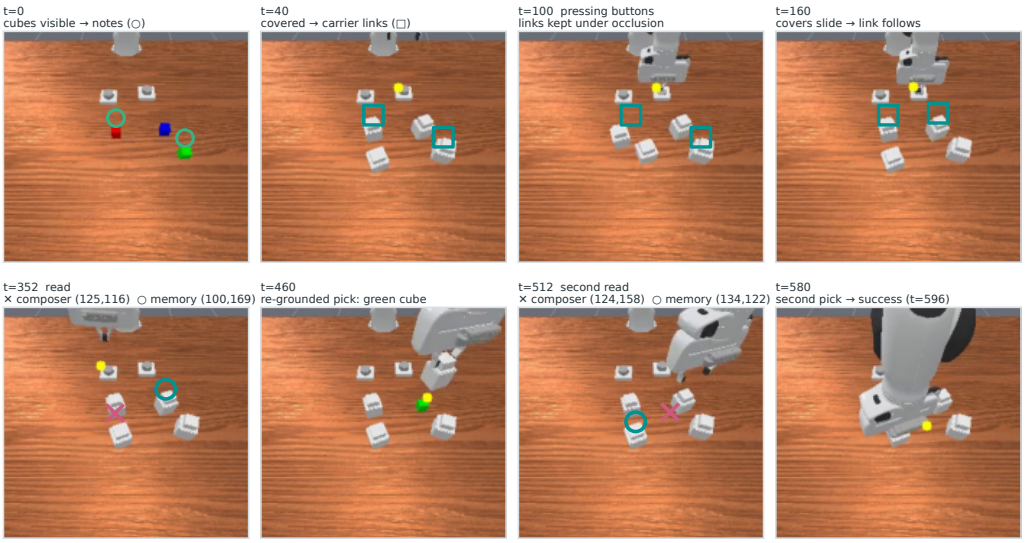


Figure 6: **A complete ButtonUnmaskSwap memory lifecycle (seed 7, episode 47).** The memory writes cube identities while they are visible, converts them to cube–cover bindings after occlusion, and updates the bindings as the covers move. At both history-dependent reads, the frozen composer proposes the wrong container; the memory retrieves the corresponding binding, re-grounds it to current geometry, and sends the corrected coordinate to the frozen VLA. The bottom lanes show the composer proposal, memory operations, and coordinate used for control.

Example 1 — ButtonUnmaskSwap (seed 7, ep. 47): live binding, cover-following, structural read, re-grounding

instruction: "first press both buttons on the table, then pick up the container hiding the green cube, finally pick up another container hiding the red cube"



memory / tool trace (from the episode log)

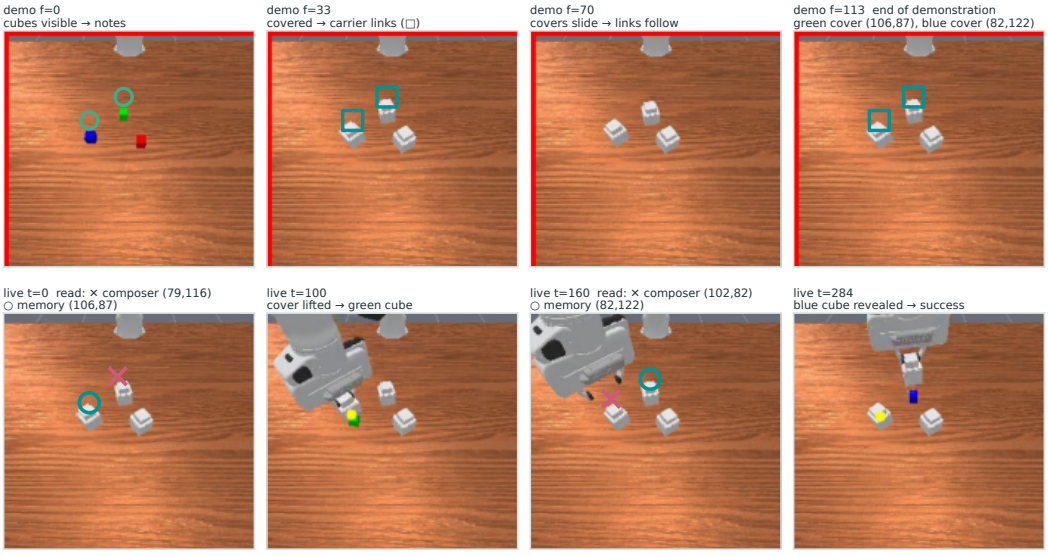
```
plan      3 votes → NEED yes · SOURCE live · WATCH green cube; red cube      compile 3 votes → [sequence] (inert: no compatible field appears)
t=0       write  green cube  object_history verified (126,189) · red cube (105,117)      evidence: detector proposal, named-colour pixels agree
t=33      green/red cube no longer at their notes → carrier links live:green cube (125,190), live:red cube (102,117) verified; DINOv2 anchor on the cover
t=120–160  covers slide: each update = nearest white-cube proposal within 18 px, DINO cosine ≥ 0.65, flow veto → link follows to (100,169)
t=352     composer 'pick up the container at <125,116> that hides the green cube' → resolve(): head 'container' ≠ note, modifier follows → READ green cube
          read_position(): current proposals near the track, identity + geometry accepted → verified → subgoal sent with (100,169)   [7 reads, t=352…448]
t=450     lift: identity/motion disagree → uncertain → reacquired → verified   (mask tool requested, unavailable on this host; identity+geometry sufficed)
t=464     'put down the container' — no history-dependent field → no read
t=512     composer <124,158> → READ red cube → (134,122)   [3 reads verified]     t=546 live:red cube → uncertain (lifted) → 3 later reads abstain, detector fallbac
t=596     success
```

Seed-7 replication of the final configuration on a local host (the SAM mask tool could not be imported there). Frames: front camera of the rollout video; the yellow dot is the coordinate handed to the VLA.

Figure 7: Live entity binding, cover following, structured access, and current-view re-grounding on ButtonUnmaskSwap.

Example 2 — VideoUnmaskSwap (seed 7, ep. 0): notes taken from the demonstration; covers followed inside it

instruction: "watch the video carefully, then pick up the container hiding the green cube, finally pick up another container hiding the blue cube"



memory / tool trace (from the episode log)

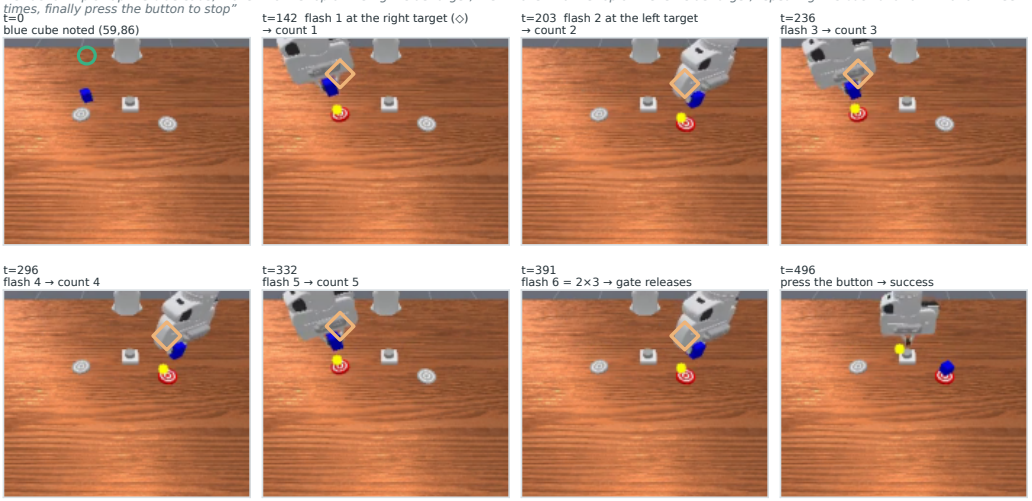
```
plan     3 votes → NEED yes · SOURCE video · WATCH blue cube; green cube        compile: 2× {} + 1× handled → no majority → no reader
demo f=0   noted blue cube and green cube (earliest pixel-verified sighting over the demonstration frames, stride 3)
demo f=33  both cubes covered → demo:blue cube carrier link verified at (106,87); demo:green cube at (82,122)
demo f=33–113  'cover of blue cube moved 42 px', 'cover of green cube moved 41 px' — followed frame by frame (DINO cosine 1.00, motion ≤ 0.1 px/obs)
demo end   demonstration read: 2/2 notes from 114 frames   ·   handled reader also wrote 'correct cube' (85,121) status uncertain — never read
live t=0   composer 'pick up the container at <79,116> that hides the green cube' → READ green cube → verified (106,87)   [7 reads t=0…96]
live t=80  demo:green cube verified → uncertain (identity/motion disagree: the container is being lifted) → reacquired (cos 0.67, 3.4 px) → (103,86)
live t=112 'put down the container' — no read
live t=160 composer <102,82> × → READ blue cube → (82,122)   [8 reads]       t=272 demo:blue cube → uncertain (lifted)
t=284    success
```

Seed-7 replication of the final configuration on a local host. Frames with a red border are demonstration frames (played before the robot acts); the memory reads them offline.

Figure 8: Demonstration-conditioned notes and cover following on VideoUnmaskSwap.

Example 3 — SwingXtimes (seed 7, ep. 1): verified flash counter and the ordinal gate

instruction: "pick up the blue cube, move it to the top of the right-side target, then move it to the top of the left-side target, repeating this back-and-forth motion three times, finally press the button to stop"



memory / tool trace (from the episode log)

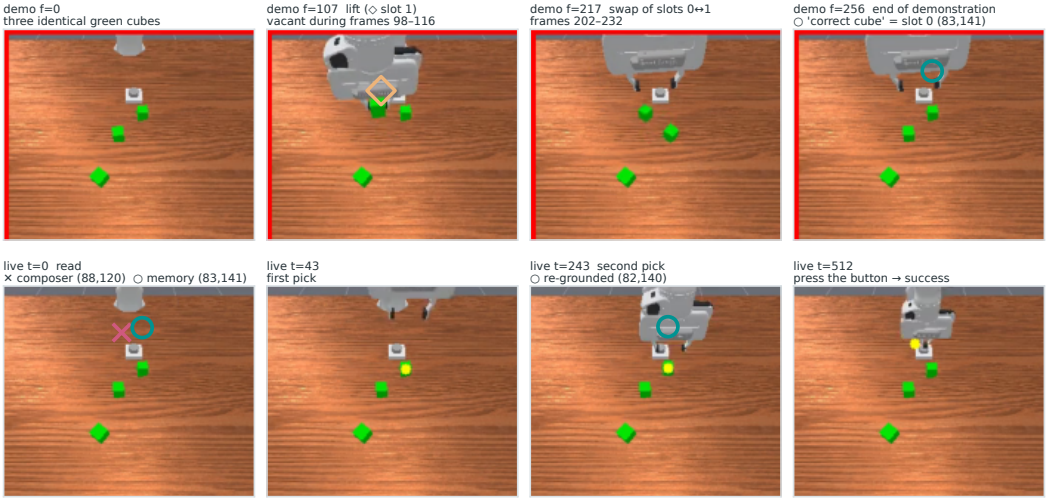
```
plan     NEED yes · SOURCE live · WATCH blue cube; left-side target; right-side target     cycle: [right-side target, left-side target] ×3, then 'press the button
compile  2× sequence + 1 invalid → [sequence] (inert)        t=0 write blue cube object_history verified (59,86)
t=144    composer 'move to the top of the left-side target … for the first time' before any flash → gate holds 'right-side target … for the first time'
t=142    flash 1 at (78,80): stationary pure-red ring, two-frame persistence, ring shape, re-entry guard → write reach_count = 1
t=192    composer 'right-side … for the second time' → flashes=1 → rewritten 'left-side … for the first time'
t=203    flash 2 (88,169) → 2      t=208,224 'left … second' → 'right … second'      t=236 flash 3 → 3      t=288 'right … third' → 'left … second'
t=296    flash 4 → 4      t=304,320 'left … third' → 'right … third'      t=332 flash 5 → 5
t=384    composer 'put the blue cube on the table' (terminal) → count 5 < 6 → held as 'left-side target … for the third time'
t=391    flash 6 → count 6 = 2 × 3 → gate releases the composer      t=464 'press the button at <61,124>'
t=496    success        (the live carrier track on the lifted cube fired too — harmless under structural access: never read)
```

Seed-7 replication of the final configuration on a local host. The red ring is the environment's reach flash; the memory counts it with a pixel predicate, never from the composer's text.

Figure 9: Verified flash counting and ordinal access on SwingXtimes.

Example 4 — VideoRepick (seed 7, ep. 0): event reader ('handled') → entity anchor → re-grounding

instruction: "watch the video carefully, then pick up the same block that was previously picked up, three times"



memory / tool trace (from the episode log)

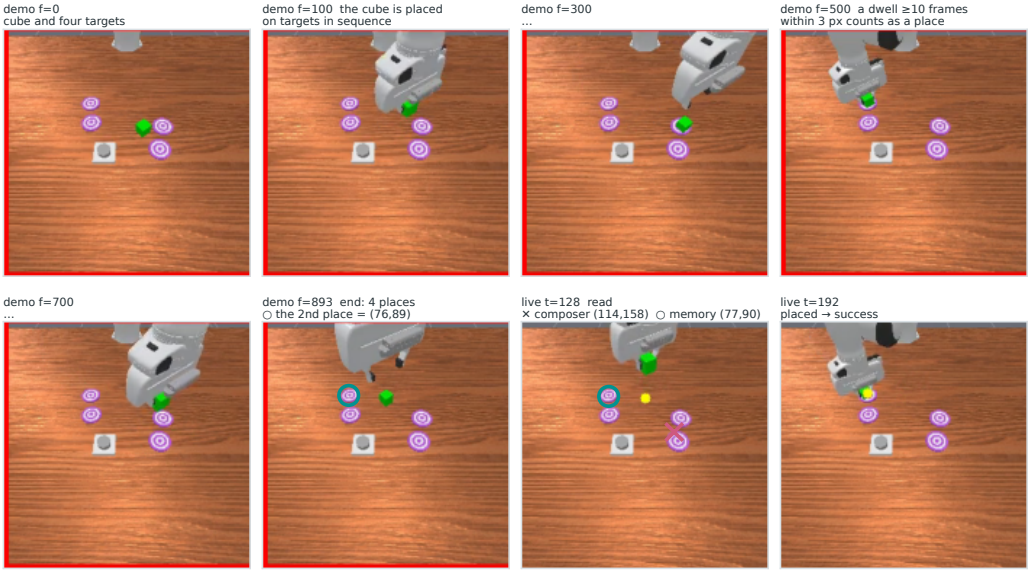
```
plan     NEED yes · SOURCE live (the agent wavers on this task) · WATCH 'the same block' — no colour word → nothing noted by the pixel verifier
compile  3× handled → reader [handled]
demo     DemoEventReader: slots = every cube blob of the first frame; events: lifted slot 1 (103,116) [frames 98–116]; swapped slots 0↔1 [frames 202–232]
         → the lifted cube ends in slot 0: 'correct cube' := (83,141)  (following one blob through the crossing would have answered (103,116))
demo     entity:correct cube seeded at its final demonstration position, DINOv2 anchor, verified
live t=0 composer 'pick up the correct cube at <88,120> for the first time' → generic gate: modifier 'correct' ∉ {colours, ordinals, sides} → READ
         read_position(): current green-cube proposals near the anchor, cos 0.99, 0.5 px → (83,141)   [8 reads t=0…112]
t=128    'put it down'      t=192 second pick → (82,140)      t=272 entity uncertain (identity/motion disagree) → t=336 reacquired (cos 0.99) → (83,140)
t=336    third pick → (83,140)      t=448 'press the button at <59,124> to finish'
t=496    success
```

Seed-7 replication of the final configuration on a local host. Demonstration frames carry a red border.

Figure 10: Demonstration event reading, entity anchoring, and re-grounding on VideoRepick.

Example 5 — VideoPlaceOrder (seed 7, ep. 1): ordinal place read from the demonstration

instruction: "watch the video carefully, then place the cube on the second target it was previously placed on"



memory / tool trace (from the episode log)

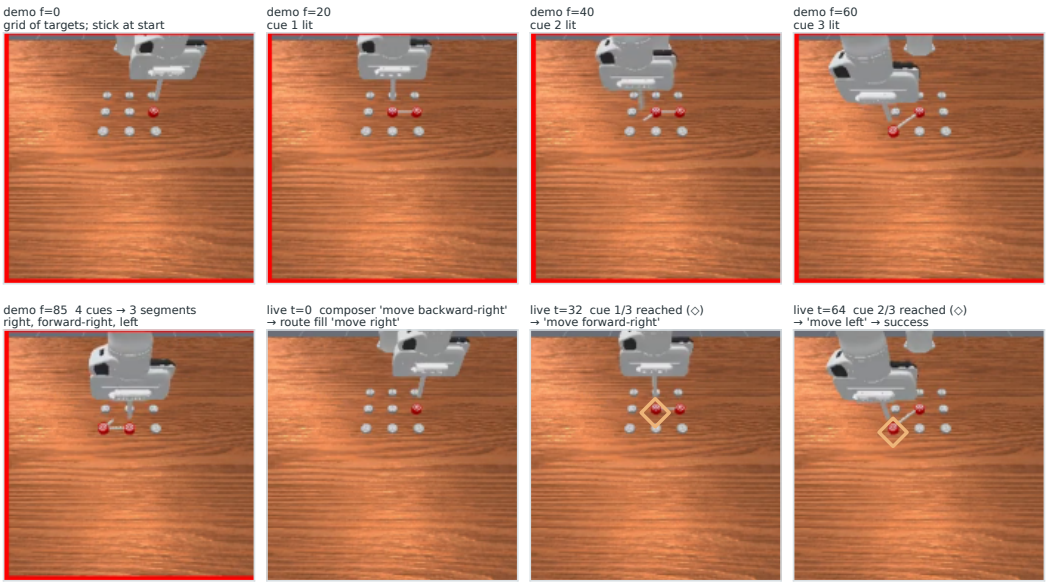
```
plan     NEED yes · SOURCE video · WATCH green cube; second target       compile 3× sequence → [sequence] (before/after reader; ordinal answered by the place list
demo     green cube noted at frame 0; covered at frame 87 → carrier link too far → note kept, status uncertain (never read)
demo     colour-blob track of the largest green blob (no jump > 30 px); dwells ≥ 10 frames within 3 px, > 10 px apart, start dropped → 4 places visited
         'the second target' → k = 2 → correct target := (76,89)   write demo_event_answer verified
route    0 cues → no route (the red-cue parser finds nothing to parse)
live t=0–112  'pick up the cube at <77,123>' → grasp verb → detector refinement → (82,128)
live t=128 composer 'place the cube onto the correct target at <114,158>' → generic gate: 'correct target' → fill (77,90)   [5 fills t=128…192]
t=192    success
```

Seed-7 replication of the final configuration on a local host. Demonstration frames carry a red border.

Figure 11: Ordinal placement state extracted from the demonstration on VideoPlaceOrder.

Example 6 — PatternLock (seed 7, ep. 0): ordered route from demonstration cues, consumed segment by segment

instruction: "watch the video carefully, then use the stick attached to the robot to retrace the same pattern"



memory / tool trace (from the episode log)

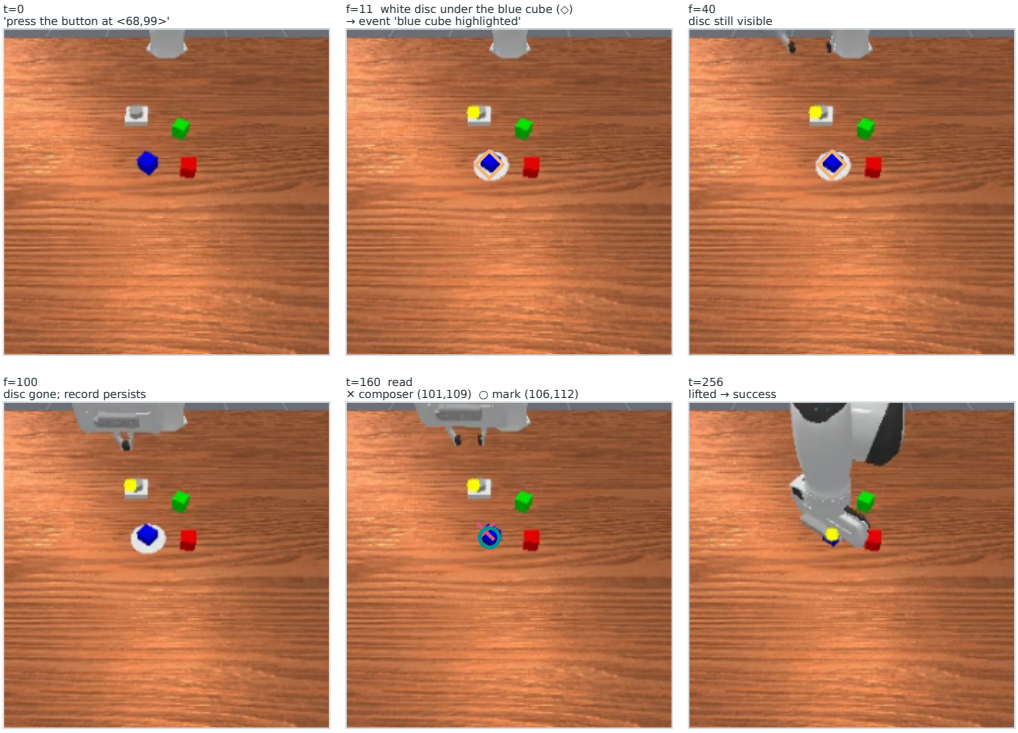
```
plan     NEED yes · SOURCE video · WATCH pattern (no colour → nothing noted; the route parser does not need a note)
demo     pure-red cue blobs per frame (r>140, g<30, b<30), merged rings, two-frame persistence → 4 cue events in order
         3 segments named from the fixed front camera: right, forward-right, left  (image-down = robot forward; side and turn sense from the tip trail)
live t=0   composer 'move backward-right' → route slot (DIR_SLOT) → filled 'move right'                [t=0,16]
live t=16  cue 1/3 reached at (84,127) (live cue tracker) → progress index 1 → 'move forward-right'      [t=32,48]
live t=48  cue 2/3 reached at (104,101) → 'move left'                                                    [t=64,80]
t=82     success   (the composer's own direction was wrong at every step; the route state supplied all three)
```

Seed-7 run with the generic gate and the same route parser/filler as the final code, before the unified entity tool stack (not needed on this task). Demonstration frames carry a red border.

Figure 12: An ordered route parsed from demonstration cues and retrieved segment by segment on PatternLock.

Example 7 — PickHighlight (seed 7, ep. 0): the mark reader keeps a transient highlight as a relation

instruction: "first press the button, then pick up all cubes that have been highlighted with white areas on the table"



memory / tool trace (from the episode log)

```
plan     NEED yes · SOURCE live · WATCH cubes (no colour → nothing noted)        readers (agent, 5 readings): mark ×3, sequence ×2 → [mark, sequence]
f=11     HighlightGraph: ≥25 bright unsaturated pixels in the 4–16 px annulus around the blue cube on 2 consecutive frames → 'blue cube highlighted' (106,112)
f=30     the disc vanishes; the relation on(blue cube, disc) is kept as marked_objects[0] = {blue, (106,112), picked=False}
t=160    composer 'pick up the highlighted cube at <101,109>, which is blue' → gate: modifier 'highlighted' → READ mark → not-yet-picked blue → (106,112)   [7 fills
t=240    mark_picked: blue colour absent from (106,112) on 5 consecutive checks → picked=True
t=256    success
```

Seed-7 run with the generic gate and the same mark reader as the final code, before the unified entity tool stack (not used on this task).

Figure 13: A transient highlight maintained as a persistent entity–mark relation on PickHighlight.