arXiv:2610.14406v1 [cs.RO] 9 Oct 2026

Memory Without Curation: Learning Multi-Stream Latent Memory for Robot Policies

Yilin Wu[1,2,†] Rutav Shah[1,3] Shuaiyi Huang[1] Sumedh A. Sontakke[1] Homanga Bharadhwaj[1,4]
Andrea Bajcsy[2] Max Simchowitz[1,2] Guanya Shi[1,2]

† work done during the internship at Amazon FAR

[1]Amazon FAR [2]Carnegie Mellon University [3]The University of Texas at Austin [4]Johns Hopkins University

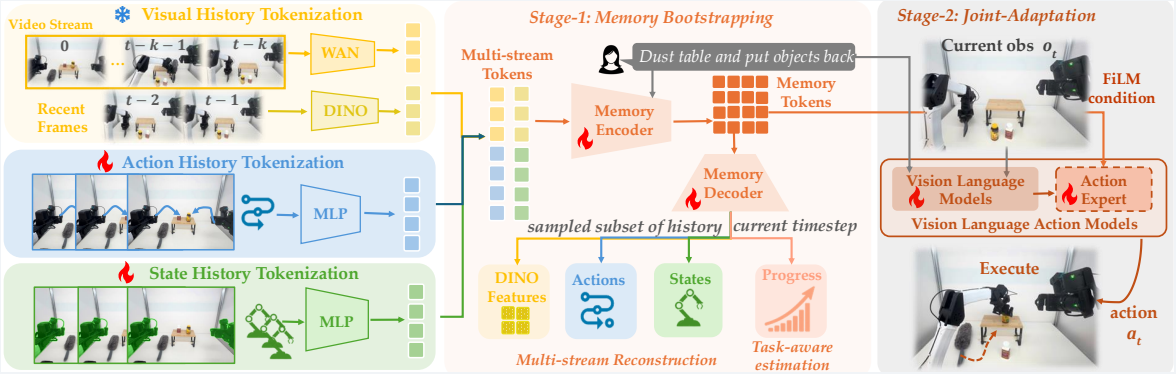


Figure 1: ***Overview of* IMPRINT.** Our latent memory pipeline learns to compress visual, action, and proprioceptive histories through multi-stream reconstruction and task-progress estimation during memory bootstrapping. It then jointly adapts with VLA for action generation.

Date: October 14, 2026
Correspondence: yilinwu@andrew.cmu.edu
Website: https://yilin-wu98.github.io/imprint_website/

Abstract

It is widely appreciated that robotic policies require long-term memory to perform complex tasks. However, directly conditioning policies on all past observations leaves policy learning susceptible to learning spurious shortcuts, and introduces large latency at policy deployment. As such, today's memory-augmented policies *curate* what to remember, either by retaining only recent frames, selected keyframes, or textual summaries, or by learning latent representations of the visual stream alone. In this work, we propose an alternative approach IMPRINT, a policy-agnostic latent memory framework that augments vision-language-action (VLA) models with multi-stream memory without explicitly constraining what the policy should remember. IMPRINT compresses different streams into a latent memory that is updated incrementally at inference, enabling low latency. To avoid shortcut learning, IMPRINT adopts a two-stage training procedure, first supervising memory to reconstruct multiple streams (observations, actions, proprioception) of past information, as well as estimate current task progress, followed by end-to-end finetuning, jointly with the policy, on action prediction. Across both simulation and real-world tasks demanding memory, IMPRINT improves average success over the base VLA by at least **26%**, outperforming popular alternatives, and maintaining **lowest inference latency** among memory-augmented VLAs. Through careful ablations, we demonstrate the utility of incorporating diverse information streams into robotic memory, validating the advantages of foregoing history curation. Project website: https://yilin-wu98.github.io/imprint_website/.

1 Introduction

Many manipulation tasks require information unavailable from current observations. For example, a robot dusting a table must move objects aside, sweep the surface, and put them back. It relies on visual, action, and proprioceptive histories to remember the objects' original locations and the direction of previous sweeping motion (Fig. 2). Because many initial vision-language-action (VLA) models [Black et al., 2024, Bjorck et al., 2025, Intelligence et al., 2025] act only on current observations, recent work has started augmenting them with memory to retain task history.

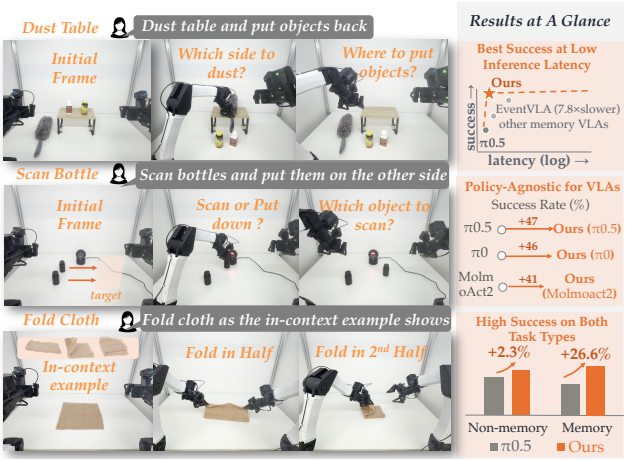


Figure 2: **Real World Tasks & Result Highlight**

To keep inference efficient, most memory-augmented policies address the challenges of history retention by *curating* what the policy should remember, either in explicit modalities like recent frames, selected keyframes, and textual summaries [Dai et al., 2026, Mark et al., 2026, Yang et al., 2026a, Zeng et al., 2026, Sridhar et al., 2026, Torne et al., 2026], or by training latent representation on carefully selected objectives [Dashora et al., 2026]. However, these choices impose a predefined notion of relevance: sparse keyframes can omit intermediate motion, textual summaries abstract away sensorimotor details, and bounded windows exclude distant history. In cloth folding (Fig. 2), visual history may capture the change of cloth shape while action and proprioceptive histories provide more information about precise grasping locations and folding trajectories, helping the robot reproduce the fold accurately. Thus, diverse memory requirements across tasks make it difficult to specify in advance exactly *what needs to be remembered*, limiting the performance of those memory-curated methods.

In this paper, we propose IMPRINT, an alternative framework that augments robotic foundation models ***with a rich multi-stream memory that learns what the policy can remember without explicitly constraining it***, while enjoying strong performance and low inference cost. To achieve this, IMPRINT compresses history of multiple *streams*, including visual observations, actions, and proprioceptive states, into a shared latent space of memory, which conditions action prediction for a policy. Building such latent memory raises two core challenges. First, *encoding*: how to compress heterogeneous and growing streams efficiently online? Second, *learning*: how to retain useful information from each stream to predict actions? Past approaches training memory representations from scratch jointly with the policy via action loss alone [Shi et al., 2026] have struggled in Sec. 5.4, because such coupling can encourage learning spurious shortcuts in demonstrations De Haan et al. [2019]. Two-stage methods instead supervise memory representations before adapting to control but typically focus on a single visual stream [Dashora et al., 2026, Koo et al., 2026].

In IMPRINT, we overcome these challenges through key innovations in both our design of memory encoding and learning. For encoding, our insight is that memory should reduce visual streams to a scale comparable to other streams, then fuse them efficiently over time. To this end, we use a pretrained video tokenizer [Wan et al., 2025] to remove spatiotemporal redundancy from visual streams, bringing them close to the scale of other streams. From there, all modalities supervise a recurrent bottleneck [Yang et al., 2025] which integrates all streams into the latent memory, updated incrementally so that efficiency is maintained as the episode grows.

For learning, our insight is that memory should be supervised directly, before adaptation to control, to retain both fine-grained multi-stream history and coarse task progress. Rather than focusing on perfect reconstruction, our goal is to "prime" the bottleneck so that end-to-end policy finetuning avoids

learning spurious memory shortcuts. Supervising on past visual features, actions, and proprioceptive states encourages memory to preserve complementary scene and motion information across all three streams, while progress estimation helps distinguish task stages that look similar but require different actions. Since neither objective requires a policy, we can have a policy-agnostic pipeline that integrates the learned memory into different VLA backbones through a lightweight adapter and jointly fine-tune memory representation with policy via action loss.

Our contributions are threefold. First, we propose IMPRINT, a policy-agnostic latent memory framework. It compresses multi-stream histories into latent memory efficiently at inference and trains with history reconstruction and progress estimation before fine-tuning it jointly with the policy. Second, we analyze how memory capacity, encoding, training objectives, and recipes affect downstream performance. Third, we evaluate IMPRINT on RMBench [Chen et al., 2026b], a customized 12-task RoboCasa benchmark [Nasiriany et al., 2024, Shah et al., 2026a] spanning memory and non-memory tasks, and three real-world tasks. IMPRINT improves success over the base VLA by **47.4% on RMBench, and 50% in the real world**, while maintaining lowest average latency of **122.8 ms** per action chunk and highest success among the memory baselines included in our latency comparison.

2 Related Work

Memory Benchmarks for Robotic Manipulation. Memory has been studied in mobile manipulation, where robots must retain spatial knowledge across environments [Zhang and Ma, 2025, Liu et al., 2025]. For tabletop manipulation, MIKASA-Robo, RMBench [Chen et al., 2026b], and RoboMME [Dai et al., 2026] probe diverse memory tasks, largely in controlled visual environments with simple objects and limited manipulation difficulty. ReMemBench [Shah et al., 2026a] and MemMimic [Gao et al., 2026] introduce more complex manipulation, but offer limited scene and background diversity. These dedicated benchmarks primarily target memory-dependent tasks; MemMimic separately evaluates non-memory RoboMimic tasks outside its suite [Gao et al., 2026]. More broadly, RoboDojo [Chen et al., 2026a] evaluates memory alongside broader manipulation capabilities, but evaluates background and scene randomization separately in its generalization tasks. Our RoboCasa-based benchmark [Nasiriany et al., 2024] jointly spans memory and non-memory tasks across diverse objects and household environments, while varying manipulation complexity.

Memory-Augmented Robot Policies. Memory has been explored across different classes of robot policies. Recurrent states and extended observation histories help visuomotor policies handle partial observability and long-horizon dependencies [Shah et al., 2026a, Torne et al., 2025, Gao et al., 2026, Jiang et al., 2026]. Structured memory methods retain object-centric scene graphs for exploration and control [Jiang et al., 2024, Qian et al., 2026, Banwasi et al., 2026]. For pretrained VLAs, existing approaches retain selected keyframes [Mark et al., 2026, Yang et al., 2026a, Shah et al., 2026b, Sridhar et al., 2026], combine short-term video memory with long-term textual summaries, [Torne et al., 2026], or learn latent memory representations [Shi et al., 2026, Koo et al., 2026]. Beyond VLAs, MemoryWAM [Yang et al., 2026b] explores persistent memory in world-action models, while MaP-WAM [Zhao et al., 2026] converts episodic language and visual memories into plans for execution. Our work focuses on memory for pretrained VLAs, studying how compact latent representations of multi-stream histories should be directly supervised and jointly adapted with control.

Learning Memory Representations. Learning compact latent memory requires supervision for what information should survive compression. Reconstruction objectives preserve information from sequential experience, including visual content [Wang et al., 2023], latent dynamics [Hafner et al., 2019], and past actions [Torne et al., 2025], while time-contrastive learning [Sermanet et al., 2018] and progress prediction [Zhai et al., 2025] provide signals about task progression. For latent memory, HAMLET [Koo et al., 2026] uses time-contrastive initialization on visual history alone, while Workspace

Models [Dashora et al., 2026] reconstruct only salient visual content. In contrast, we supervise memory on all three streams, reconstructing visual, action, and proprioceptive histories together with task progress. Rather than aiming for faithful reconstruction, this supervision *primes* the bottleneck, so that joint fine-tuning with the policy is less likely to learn spurious memory shortcuts from demonstrations.

3 Problem Formulation

Notation & Setup. At any real time t, the robot observes RGB camera images $I_t \in \mathscr{I}$ from cameras and proprioceptive states $q_t \in \mathscr{Q}$, which we concatenate as $o_t := [I_t, q_t]$. Let $a \in \mathscr{A}$ denote a low-level action (e.g., joint states), $\mathbf{a}_t := a_{t:t+h}$ as an h-step action chunk, and ℓ as the task description. In this paper, we focus on VLAs with parameters θ that acts only on the current observation, $\mathbf{a}_t \sim \pi_\theta(\cdot \mid o_t, \ell)$. We denote the history by $\mathrm{H}_t := \big(o_{\le t},\ a_{<t}\big)$ which contains all past visual observations, proprioceptive states, and actions. $\mathbb{D}$ is a set of demonstrations to train policies.

Memory-Augmented Policies. Many manipulation tasks require information beyond current observations: two histories $\mathrm{H}_t \neq \mathrm{H}'_t$ can end at the same observation o_t yet call for different actions, $\pi(\mathbf{a}_t \mid o_t, \ell, \mathrm{H}_t) \neq \pi(\mathbf{a}_t \mid o_t, \ell, \mathrm{H}'_t)$. For example, while a robot dusts a table, the current scene no longer shows original locations or directions to sweep, but returning the objects and finishing the dusting require that. To handle such *history-dependent* steps, a memory-augmented policy $\pi^{\mathrm{mem}}_\theta$ usually summarize the history into a memory representation m_t through a function f and conditions on it to generate actions: $m_t = f(\mathrm{H}_t, \ell), \mathbf{a}_t \sim \pi^{\mathrm{mem}}_\theta(\cdot \mid o_t, \ell, m_t)$.

The central question is therefore how to design the memory function f without restricting what it can represent, as explicit forms do. Since H_t grows continuously and spans heterogeneous visual, action, and proprioceptive streams, retaining it in full is neither scalable nor necessary for control. Instead, f should compress the growing history into a bounded representation that preserves the information needed for future decisions. We therefore consider the setting where f is a learned latent memory encoder $\mathrm{E}^{\mathrm{mem}}_\phi$ that maps the multi-stream interaction history to a continuous representation: $m_t = \mathrm{E}^{\mathrm{mem}}_\phi(\mathrm{H}_t, \ell) \in \mathbb{R}^{B\times d}$, where B does not grow with t. Our goal is a memory representation m_t that compresses the multi-modal history context H_t without changing the action distribution:

$$\pi(\mathbf{a}_t \mid o_t, \ell, \mathrm{H}_t) = \pi(\mathbf{a}_t \mid o_t, \ell, m_t). \tag{3.1}$$

To approach this goal, the encoder and policy are learned from demonstrations by minimizing an objective L that depends on the memory:

$$\min_{\theta,\phi}\ \mathbb{E}_{\mathbb{D},t}\Big[\mathrm{L}\big(\mathrm{H}_t, \mathbf{a}_t;\ \pi_\theta,\ \mathrm{E}^{\mathrm{mem}}_\phi\big)\Big]. \tag{3.2}$$

Achieving (3.1) through (3.2) raises two challenges: how to design the encoder $\mathrm{E}^{\mathrm{mem}}_\phi$, and how to choose the objective L.

Challenge 1: Encoding Heterogeneous Interaction Histories Online. The history combines a high-dimensional video stream with low-dimensional proprioceptive states and actions, which differ in scale and structure. The video stream is particularly hard to compress: both spatial and temporal dimensions have high redundancy, yet the small changes between them, such as object locations, matter for later decisions. Moreover, the history grows at every step while memory is queried at every control step, so re-encoding the full history adds latency that grows with the episode length.

Challenge 2: Choosing the Memory Objective. The standard choice for Eq. 3.2 is end-to-end optimization through action loss [Shi et al., 2026], where the memory is learned only through gradients backpropagated from the policy:

$$\mathrm{L}_{\mathrm{act}}(\theta,\phi) = \mathbb{E}_{\mathbb{D},t}\Big[-\log \pi^{\mathrm{mem}}_\theta\big(\mathbf{a}_t \mid o_t, \ell, \mathrm{E}^{\mathrm{mem}}_\phi(\mathrm{H}_t, \ell)\big)\Big]. \tag{3.3}$$

This objective supervises memory only indirectly through action prediction, coupling what the bottleneck should retain with how the policy uses it for control. In particular, it provides no explicit supervision that memory preserves information from each heterogeneous history stream.

4 IMPRINT: Interaction Memory with Progress- and Reconstruction-Informed Training

We address the challenges in Sec. 3 with efficient encoding and two-stage learning. We first compress visual, action, and proprioceptive histories into a fixed-size latent memory (Sec. 4.1). We then bootstrap memory with history reconstruction and progress estimation (Sec. 4.2.1), before jointly adapting the memory and VLA with action supervision for control (Sec. 4.2.2).

4.1 Multi-stream Memory Encoding

The first challenge lies in the efficiency to compress a growing, heterogeneous interaction history into latent memory. We address this with two levels of compression: within stream and across streams over time. Because video dominates the history in both dimensionality and redundancy, we first compress it with a frozen causal video tokenizer [Wan et al., 2025, Agarwal et al., 2026]. The memory encoder then integrates the resulting visual tokens with actions and proprioceptive states over time into a fixed number of latent memory tokens as shown in left panel of Fig. 1.

Multi-Stream Memory Tokenization. We represent history through four token streams: *historical video*, *recent images*, *executed actions*, and *proprioceptive states*. Let $I_{\leq n_t}$ denote the previous observed frames available at time t. With L frames per chunk, $K_t = \lfloor n_t/L \rfloor$ chunks are complete. A frozen causal video tokenizer $\mathrm{T}_\psi^{\mathrm{vid}}$ [Wan et al., 2025] divides these frames $I_{\leq K_t L}$ into chunks and then tokenize them into a small set per camera. The remaining frames $I_{K_t L+1:n_t}$ form the recent-image stream, tokenized by off-the-shelf DINOv3 encoder T_ψ^{v} [Siméoni et al., 2025] to construct visual features z_k^v. Separate Multi-Layer Perceptrons (MLPs) embed previously executed actions $a_{<t}$ and proprioceptive states $q_{\leq t}$ into the other two streams.

Memory Bottleneck. Let e_t^{vid}, e_t^{recent}, e_t^{act}, and e_t^{state} denote the outputs of the four tokenization streams, and let $e_t = (e_t^{\mathrm{vid}}, e_t^{\mathrm{recent}}, e_t^{\mathrm{act}}, e_t^{\mathrm{state}})$ denote their concatenation.We introduce B learned query embeddings that read out information from the four token streams conditioned on the task description ℓ, following [Jaegle et al., 2021]. The memory encoder computes $m_t = \mathrm{E}_\phi^{\mathrm{mem}}(e_t; \ell) \in \mathbb{R}^{B\times d}$. The learned queries are shared across episodes, and together with language features of ℓ, compress the history into B memory tokens. We consider two bottlenecks with the same supervision (Sec. 4.2). Writing the four token streams collectively as e_t, they compute

$$\text{Full Attention:} \quad m_t = \mathrm{E}_\phi^{\mathrm{mem}}(e_t; \ell), \tag{4.1}$$

$$\text{Recurrent:} \quad (m_t, h_t) = \mathrm{E}_\phi^{\mathrm{mem}}(e_t; h_{t-1}, \ell). \tag{4.2}$$

i) *Full Attention bottleneck.* Following Perceiver [Jaegle et al., 2021] and Workspace Models [Dashora et al., 2026], the queries cross-attend to all history tokens, including the recent window. Each query revisits the full history, so storage and per-query cost grow linearly with history length.

ii) *Recurrent bottleneck.* We use gated delta (GDN) blocks [Yang et al., 2025] to process each chunk e_t by updating fixed-size matrix states, collectively denoted by h_t. The queries read these states and then attend to the recent window, keeping storage and per-query cost bounded independently of history length. Training uses the parallel formulation of the gated delta blocks.

4.2 Decoupled Memory Supervision

Optimizing Eq. 3.2 requires learning both what information memory should retain and how the policy should use it for control. With action loss alone, the memory encoder is guided only indirectly through action prediction, providing no explicit requirement that it preserve information from each history stream. We therefore decouple these two roles: first learn memory with direct, policy-independent supervision, and then jointly adapt the memory and policy with action loss as in Fig. 1.

Memory Objective & Training Pipeline. We first learn latent memory with two complementary self-supervised objectives that force it to capture both information from interaction histories and global task context. *History reconstruction* requires m_t to reconstruct past visual features, actions, and proprioceptive states, to retain information from each stream. However, reconstruction alone does not explicitly capture the task stage. We therefore use *progress estimation*, to encourage m_t to predict task progress and thereby encode global temporal context. Both objectives are policy-independent and annotation-free, implemented with a decoder $\mathscr{D}_\varphi$ that operates on m_t.

Once memory representations are learned, we adapt them for control by jointly training the memory encoder and policy via the action-prediction objective, leading to the two-stage training procedure:

$$\begin{aligned} &\text{Stage 1:} \quad \bar{\phi}, \bar{\varphi} = \arg\min_{\phi,\varphi} \mathbb{E}_{\mathbb{D},t}\Big[\mathrm{L}_{\mathrm{mem}}\Big(\mathscr{D}_\varphi\big(\mathrm{E}_\phi^{\mathrm{mem}}(\mathrm{H}_t,\ell)\big),\, \mathbf{y}_t\Big)\Big], \\ &\text{Stage 2:} \quad \theta^\star, \phi^\star = \arg\min_{\theta,\phi} \mathbb{E}_{\mathbb{D},t}\Big[-\log \pi_\theta\big(\mathbf{a}_t \mid o_t, \ell, \mathrm{E}_\phi^{\mathrm{mem}}(\mathrm{H}_t,\ell)\big)\Big], \quad \phi^{(0)} = \bar{\phi} \end{aligned} \tag{4.3}$$

where $\mathbf{y}_t$ represents the reconstruction targets at sampled past steps and the progress ρ_t, and $\mathrm{L}_{\mathrm{mem}}$ is the prediction error defined in (4.4). Stage 1 trains the encoder $\mathrm{E}_\phi^{\mathrm{mem}}$ and decoder $\mathscr{D}_\varphi$ without the policy. Stage 2 discards the decoder and initialize encoder $\mathrm{E}_\phi^{\mathrm{mem}}$ from parameters $\bar{\phi}$ and finetunes the VLA policy π_θ jointly with the memory encoder, so that the action objective adapts latent memory toward decision-relevant content and teaches the policy to read it.

4.2.1 Stage 1: Memory Pre-training

To use for multi-stream memory, the decoder $\mathscr{D}_\varphi$ reconstructs K historical steps sampled uniformly at each timestep t [Jaegle et al., 2021]. For each sampled step $k \le t$, it forms one query per visual patch and one summary query, all using the embedding for k from a learned timestep table shared across episodes. These queries attend to each other within the step and cross-attend to m_t. The patch queries predict that step's visual features $\hat{z}_k^v$, while the summary query predicts its action $\hat{a}_k$ and state $\hat{q}_k$. A separate global query predicts progress $\hat{\rho}_t$. The loss is:

$$\mathrm{L}_{\mathrm{mem}} = \underbrace{\frac{1}{K}\sum_k \Big[\lambda_v \big\|\hat{z}_k^v - z_k^v\big\|^2 + \lambda_a \big\|\hat{a}_k - a_k\big\|^2 + \lambda_q \big\|\hat{q}_k - q_k\big\|^2\Big]}_{\text{history reconstruction}} + \underbrace{\lambda_\rho\big(\hat{\rho}_t - \rho_t\big)^2}_{\text{progress estimation}}, \tag{4.4}$$

where $z_k^v = \mathrm{T}_\psi^v(I_k)$ are frozen, normalized patch features and $\rho_t \in [0,1]$ is the normalized progress of the demonstration (i.e. current time-step over total episode length). The first three terms supervise history reconstruction at individual past steps, and the last term supervises progress estimation over the entire history, thereby integrating all relevant streams. These objectives aim to learn useful memory representations rather than achieve perfect compression. Together, they *prime* memory tokens with complementary scene, motion, and task-progress information for downstream policy fine-tuning, helping reduce reliance on spurious memory shortcuts in the demonstrations.

4.2.2 Stage 2: Policy-Agnostic Memory Adaptation

Memory Interface. Stage 1 produces a policy-independent memory encoder that can integrate with different VLAs through a lightweight action-expert adapter, following [Dai et al., 2026] (Fig. 1). At each layer, action tokens x cross-attend to m_t and receive FiLM modulation [Perez et al., 2018]: $\tilde{x} = \mathrm{FiLM}\big(x,\ \mathrm{CrossAttn}(x, m_t)\big)$. This preserves the VLA's native inputs, context length, and conditioning, while better preserving pretrained representations than appending memory to the VLM context [Dai et al., 2026]. Attending to a fixed number of memory tokens adds little cost, independent of history length. We instantiate this adapter in different VLA backbones including π_0 [Black et al., 2024], $\pi_{0.5}$ [Intelligence et al., 2025], and MolmoAct2 [Fang et al., 2026] (App. A.1.4).

Joint Adaptation. Following Stage 2 of (4.3), we discard the decoder and initialize the policy from θ_0 and the encoder from $\bar{\phi}$. The adapter is initialized near zero, so training starts close to the original VLA. We then optimize the action objective (3.3) with the VLA's native loss (e.g., flow matching), updating the encoder, adapter, and VLA. Because the learned latent memory already carries critical information from the history, the policy can easily rely on that to generate correct actions, and the action gradient in turn further refines memory for control.

5 Experiments

We evaluate whether IMPRINT improves manipulation tasks with different memory requirements while keeping memory compact and efficient (full details in App. B to E). We first compare IMPRINT against prior memory methods in simulation and real world, on task success and inference latency in Sec. 5.2. We then analyze *memory encoding*, how memory capacity and recurrent compression affect memory representation and performance in Sec. 5.3; and *memory learning*, how memory supervision and adaptation affect control, and its flexibility to different VLAs in Sec. 5.4.

5.1 Experimental Setup

Benchmarks and Tasks. We evaluate IMPRINT on two simulation benchmarks and three real-world manipulation tasks. In simulation, we use all 9 memory-dependent tasks from RMBench [Chen et al., 2026b] and a customized 12-task benchmark built on RoboCasa [Nasiriany et al., 2024] and ReMemBench [Shah et al., 2026a], containing 6 memory and 6 non-memory tasks with diverse objects and scenes. This setup evaluates both the benefits of memory and its impact on tasks without memory necessity. In the real world, we evaluate three memory-dependent tasks on a bimanual YAM platform: *Dust Table*, *Scan Bottle*, and *Fold Cloth*, covering spatial memory of object locations, task-progress tracking of scanning, and learning from in-context robot demonstrations of varied folding styles. Task details are provided in Appendix B.1, B.2, B.3.

Baselines. We compare **IMPRINT** against three VLAs, π_0, $\pi_{0.5}$, MolmoAct2 without memory and three categories of memory-augmented VLAs from prior work: sampling visual frames with heuristics **Uniform Sampling** and **EventVLA** [Dai et al., 2026, Yang et al., 2026a], hierarchical memory methods with textual summary and keyframes **Mem-0** and **MemER** [Chen et al., 2026b, Sridhar et al., 2026], and latent-memory methods **HAMLET** and **MemoryVLA** [Koo et al., 2026, Shi et al., 2026]. We also conduct controlled ablations of encoder, memory capacity, training objectives, and adaptation with VLAs. Details of baselines and ablations are in Table. 5, App. A, C.

Training and Evaluation. Unless otherwise specified, we use $\pi_{0.5}$ as base policy and $B = 256$ memory tokens with $d = 1024$. We report results over 100 trials per task in RmBench and 50 trials per task in RoboCasa. Following [Team et al., 2025], we plot the smoothed distribution of task success symmetrically, with width indicating density and a horizontal line marking the mean. Implementation

details, full breakdown of per-task results, additional ablations, and detailed failure analysis are in App. A, C, D, and E.

5.2 IMPRINT Outperforms Existing Memory-Augmented VLAs with Better Task Performance and more Efficient Inference

We first evaluate the overall effectiveness of IMPRINT across two simulation benchmarks and real-world manipulation tasks (App. C.1). We compare against the base VLA and existing heuristic, hierarchical and latent memory approaches, measuring both task success and inference latency.

Simulation Results. Figure 3 compares IMPRINT with heuristic, hierarchical, and latent memory approaches on 9 memory-dependent RMBench tasks. Heuristic selection or sampling of sparse visual frames helps over the no-memory $\pi_{0.5}$ baseline from 25.4% to 67.8% but fails on tasks that need denser information (e.g., press button). Hierarchical methods achieve at most 42.0% success, caused by the loss of information in the reduced subset of memories and large action jump near the subtask boundaries. Prior latent memories achieve at most 41.7%, but either operate primarily on visual history or lack direct supervision to preserve information from each history stream. In contrast, recurrent IMPRINT supervises a multi-stream latent memory to retain visual, action, and proprioceptive history while tracking task progress, reaching 72.8% success.

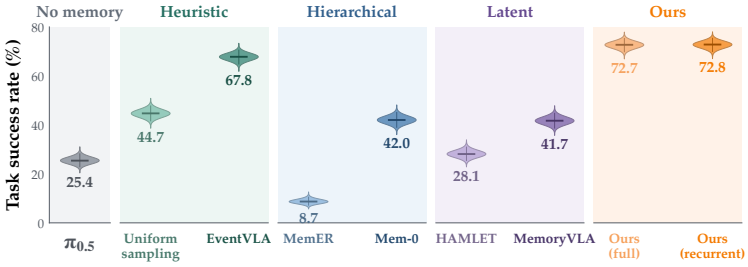


Figure 3: **Average Task Success on Rmbench**

Figure 4 examines whether IMPRINT helps with tasks that do not require history. Heuristic memory can improve memory-dependent performance about 15.3% but substantially degrades non-memory tasks by 22%, indicating that simply exposing the VLA to additional history can interfere with behaviors already solvable from the current observation. In contrast, recurrent IMPRINT raises memory task success to 65.3% while slightly improving non-memory task performance to 68.0%, achieving 66.7% overall. This suggests that compressing history into a learned latent representation allows the policy to exploit memory when needed without harming non-memory tasks.

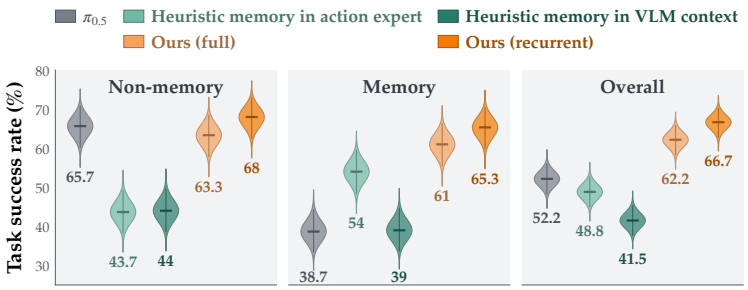


Figure 4: **Average Task Performance For Memory and Non-Memory Tasks in Robocasa**

Real-World Results. Figure 5 shows the gains of IMPRINT transfer to real-world tasks. IMPRINT achieves 80%, 75%, and 72% success across three tasks, consistently outperforming all baselines. Compared with the strongest baseline (uniform sampling), IMPRINT improves average success by 26.7%. The consistent gains over all

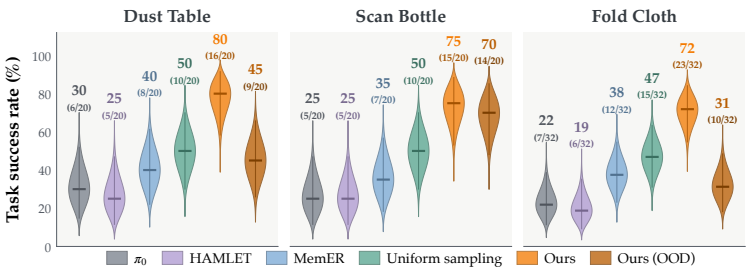


Figure 5: **Real World Task Performance Across 3 Tasks**

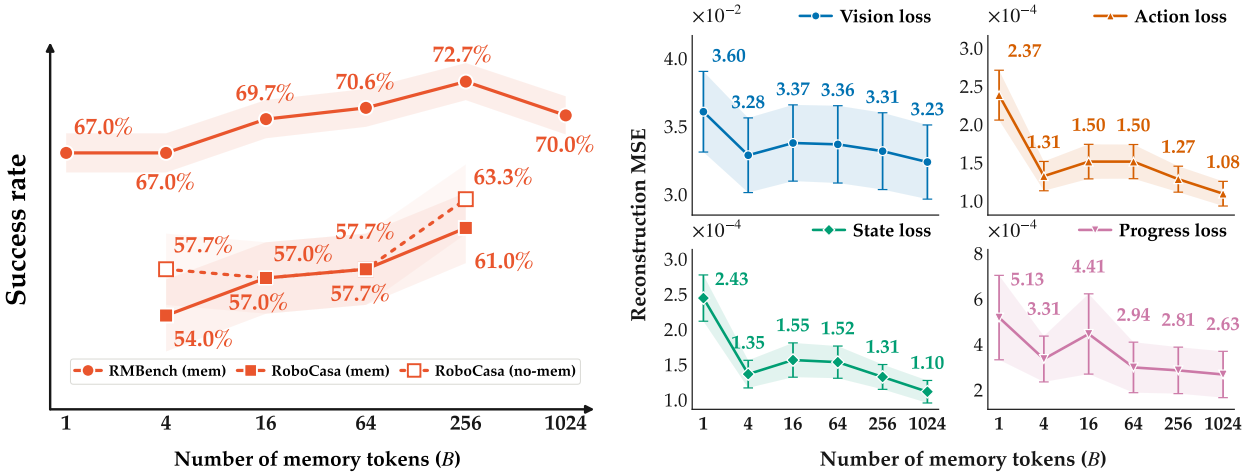


Figure 7: **Ablations on Memory Capacity.** We evaluate IMPRINT trained with different memory tokens against task performance (left) and supervision loss evaluated on the validation set (right).

memory baselines suggest that directly learning a compact multi-stream memory is also useful under real-world tasks that requires spatial memory, progress estimation of scanning, and in-context learning of folding styles.

Real-World OOD Evaluation. We further evaluate IMPRINT on *unseen* objects (App. B.5.3), achieving success rates of 45%, 70%, and 31% across three tasks (Fig. 5). These results provide encouraging initial evidence that the learned memory can transfer beyond the training configurations. While generalization remains challenging for longer-horizon, precision-sensitive tasks such as *Dust Table* and *Fold Cloth*, this OOD evaluation highlights the potential of IMPRINT to handle novel objects.

Success and Inference Efficiency. Figure 6 compares task success and average inference latency per 50-step action chunk under L40S GPU. Prior memory methods incur either limited gains or substantial latency: HAMLET and MemoryVLA reach 28.1% and 41.7% success at 144 and 321 ms, while EventVLA reaches 67.8% but requires 960 ms due to long context to Qwen backbone. Hierarchical methods are even slower with several seconds from high-level VLMs. In contrast, recurrent IMPRINT achieves the highest success (72.8%) at 122.8 ms, closest to the no-memory $\pi_{0.5}$ runtime and dominating all evaluated memory baselines in both success and latency. Full inference time analysis is in App. C.5.

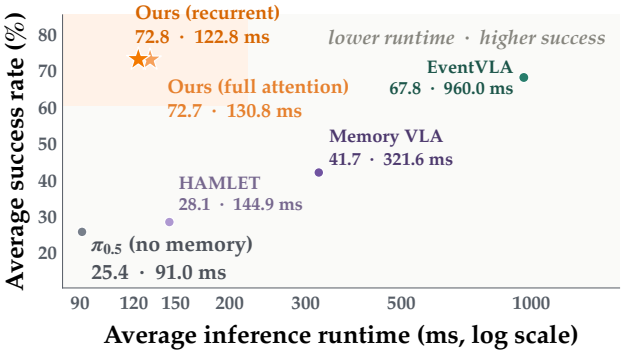


Figure 6: **Runtime vs Success on RMBench**

5.3 Richer Multi-Stream Inputs, Greater Capacity, and Recurrent Compression Enable More Effective Memory Encoding

We next study the encoding challenge from Sec. 3: compressing a growing interaction history into a continuous latent representation. We examine three properties of the memory bottleneck: its *input*, which evaluates whether multi-stream information is necessary for the task; its *capacity*, which controls how much historical information can be retained, and its *update mechanism*, which determines how efficiently history is accumulated over time. Detailed results are in App. C.2.

Multi-Stream History Provides Complementary Information. We ablate the different input streams of histories supplied to memory while holding the full attention encoder and two-stage training recipe fixed. In Fig. 8 (right), Vision alone achieves 60.6% success; adding action history improves performance to 68.4%, while the visual-and-state variant reaches 60.9%. IMPRINT which combines all streams of histories performs best at 72.7%, showing that those streams provide complementary information for memory-dependent control and specific curation of memory to certain stream, e.g., vision, can decrease the task performance.

More Capacity Preserves More Useful Context. In Fig. 7, with the full-attention encoder and training recipe fixed, increasing memory capacity from 1 to 256 tokens raises RMBench success from 67.0% to 72.7%, and increasing it from 4 to 256 tokens raises RoboCasa memory-task success from 54.0% to 61.0%. Non-memory success stays near 57% through 64 tokens before reaching 63.3% at 256. However, 1024 tokens yields the lowest reconstruction losses but lowers RMBench success to 70.0%, and increasing from 4 to 16 tokens improves success despite higher reconstruction errors, suggesting that reconstruction serves to *prime* useful memory representations for policy adaptation rather than to achieve perfect compression.

Recurrent Compression Improves Encoding Efficiency. Beyond capacity, the memory encoder must encode a growing history efficiently. At 256-token capacity, our recurrent encoder matches or improves the full attention variant's performance while reducing average inference latency from 130.8 ms to 122.8 ms per action chunk. Both achieve approximately 72.8% success on RMBench (Fig. 3), while recurrent encoding improves overall RoboCasa success from 62.2% to 66.7% (Fig. 4).

5.4 Memory Pretraining and Joint Adaptation Improve Task Performance

Pretraining and Joint Adaptation Are Both Important. We compare learning memory end-to-end via action supervision (E2E) in Eq. 3.3, frozen pretrained memory without adaptation, and pretraining followed by joint memory–VLA adaptation. In Fig. 8 (left), success improves from 50.8% (E2E) to 58.8% with bootstrapping alone, and further to 72.7% with joint adaptation. This shows that both initial memory learning and subsequent adaptation for action prediction are important.

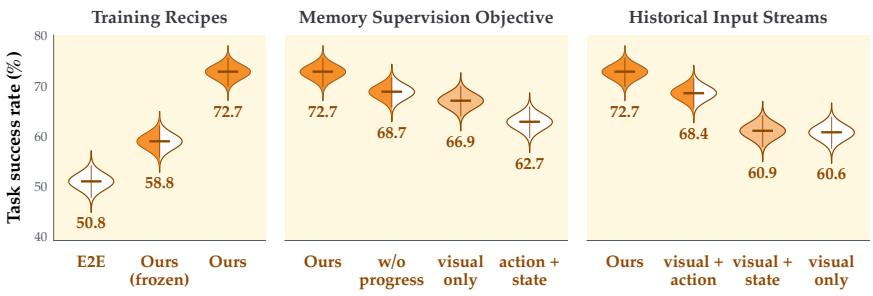


Figure 8: **Ablations of Memory Learning.** We ablate different input streams (right) and varied supervision objectives (middle) and training recipes (left) of IMPRINT on RMBench.

Two-stage Training Produces More Informative Memory. Figure 9 examines what memory retains and its impact on task success on *Cover Block* (App. C.4). Memory pretraining improves linear-probe accuracy of classifying the color order by 63.4%, reaching 91.5% after joint adaptation, while IMPRINT improves T-SNE structure to $S = 0.36$ with better class separation. Attention from both frozen and IMPRINT focus on the initial frames of head camera with color order revealed, whereas E2E focus on state history. However, even with right attention, only joint adaptation enables IMPRINT to uncover blocks in the correct order, showing pretraining improves retention of critical information while adaptation optimizes it for action prediction.

Reconstruction and Progress Estimation are Complementary Objectives. We next ablate different supervision objectives with the fixed adaptation in Fig. 8 (middle). The combination of visual, action, and state reconstruction with progress estimation obtains 72.7% success. Removing progress estimation reduces success to 68.7%, while using only visual reconstruction or only action and state

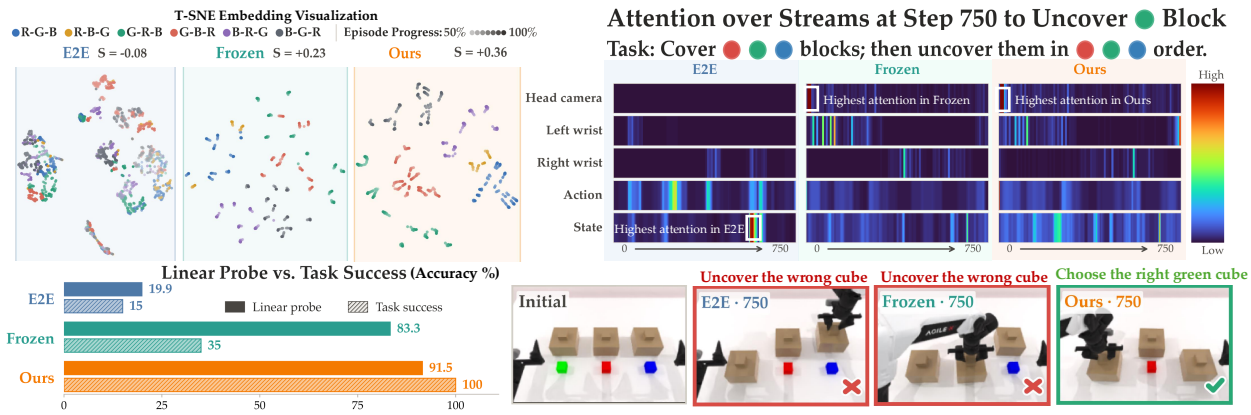


Figure 9: **Analysis of Memory Bottleneck.** We compare E2E, frozen memory encoder, and our adapted memory with t-SNE embeddings, linear probes, and attention visualization on *Cover Block*.

reconstruction further decreases success to 66.9% and 62.7%, respectively. Memory thus benefits from both history retention and progress estimation.

IMPRINT is Policy-agnostic across VLAs. Because Stage 1 learning is independent of the downstream policy, we apply the same memory pipeline and adaptation procedure to three different VLA backbones. In Fig. 10, memory augmentation improves success from 25.4% to 72.7% for $\pi_{0.5}$, from 17.4% to 63.0% for π_0, and from 18.8% to 59.7% for MolmoAct2. The consistent gains across different base policies show that the proposed memory bootstrapping and joint-adaptation recipe is not tied to a particular VLA. Details are in App. A.1.4 and Table 16.

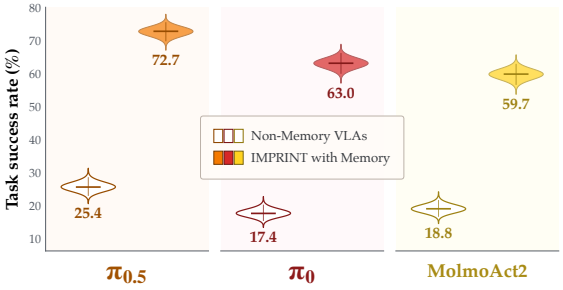


Figure 10: **IMPRINT Improves Different VLA Backbones with Memory.**

6 Conclusion and Limitations

We presented IMPRINT, a latent memory framework that augments VLAs with multi-stream memory without constraining what the policy should remember. IMPRINT compresses visual, action, and proprioceptive histories into a latent memory, supervised with reconstruction and progress estimation before joint fine-tuning with VLAs. Across simulation, real-world, and three VLA backbones, IMPRINT outperforms base VLAs and other memory approaches with modest inference overhead.

Limitations. Latent memory is less interpretable than explicit memory, making failure analysis more challenging. Extending the memory to include failures for in-context learning could further broaden IMPRINT applicability. Currently, IMPRINT learns memory representations solely from teleop. demonstrations; incorporating more scalable data sources is an exciting future direction.

7 Acknowledgments

We thank Yihui Gao and Zhanyi Sun for valuable discussions on memory learning during the early stages of brainstorming for this work. We also thank Zhenyu Jiang and Huihan Liu for sharing insights into real-world robotic tasks and the importance of memory. We are grateful to Chaoyi Pan for assistance with the research infrastructure, and to Haoru Xue and Tonghe Zhang for helpful discussions on in-context learning.

References

Niket Agarwal, Arslan Ali, Jon Allen, Martin Antolini, Adeline Aubame, Alisson Azzolini, Junjie Bai, Maciej Bala, Yogesh Balaji, Josh Bapst, et al. Cosmos 3: Omnimodal world models for physical ai. *arXiv preprint arXiv:2606.02800*, 2026.

Anuva Banwasi, William Muckelroy III, Priya Sundaresan, Linfeng Zhao, Jeannette Bohg, and Cherie Ho. Messymem: Learning-from-doing memory for mobile manipulation. *arXiv preprint arXiv:2609.15976*, 2026.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. pi: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Tianxing Chen, Yue Chen, Zixuan Li, Junyuan Tang, Kailun Su, Weijie Wan, Baijun Chen, Haoran Lu, Haowen Yan, Honghao Su, et al. RoboDojo: A unified sim-and-real benchmark for comprehensive evaluation of generalist robot manipulation policies. *arXiv preprint arXiv:2607.04434*, 2026a.

Tianxing Chen, Yuran Wang, Mingleyang Li, Yan Qin, Hao Shi, Zixuan Li, Yifan Hu, Yingsheng Zhang, Kaixuan Wang, Yue Chen, et al. Rmbench: Memory-dependent robotic manipulation benchmark with insights into policy design. *arXiv preprint arXiv:2603.01229*, 2026b.

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. *Proceedings of the 43 rd International Conference on Machine Learning (ICML)*, 2026.

Nitish Dashora, Douglas Chen, Idan Shenfeld, John Marangola, Pulkit Agrawal, and Max Simchowitz. Workspace models: Lightweight robotic memory via saliency-driven supervision, 2026. URL https://arxiv.org/abs/2609.20820.

Pim De Haan, Dinesh Jayaraman, and Sergey Levine. Causal confusion in imitation learning. *Advances in neural information processing systems*, 32, 2019.

Haoquan Fang, Jiafei Duan, Donovan Clay, Sam Wang, Shuo Liu, Weikai Huang, Xiang Fan, Wei-Chuan Tsai, Shirui Chen, Yi Ru Wang, et al. Molmoact2: Action reasoning models for real-world deployment. *arXiv preprint arXiv:2605.02881*, 2026.

Yihuai Gao, Jinyun Liu, Shuang Li, and Shuran Song. Gated memory policy. *arXiv preprint arXiv:2604.18933*, 2026.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. pi 0.5: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pages 4651–4664. PMLR, 2021.

Hanxiao Jiang, Binghao Huang, Ruihai Wu, Zhuoran Li, Shubham Garg, Hooshang Nayyeri, Shenlong Wang, and Yunzhu Li. Roboexp: Action-conditioned scene graph via interactive exploration for robotic manipulation. In *Conference on Robot Learning (CoRL)*, 2024.

Yunfan Jiang, Yevgen Chebotar, Ruijie Zheng, Fengyuan Hu, Yunhao Ge, Jimmy Wu, Tianyuan Dai, Scott Reed, Li Fei-Fei, Yuke Zhu, and Linxi "Jim" Fan. Robottt: Context scaling for robot policies. *arXiv preprint arXiv: 2607.15275*, 2026.

Myungkyu Koo, Daewon Choi, Taeyoung Kim, Kyungmin Lee, Changyeon Kim, Younggyo Seo, and Jinwoo Shin. Hamlet: Switch your vision-language-action model into a history-aware policy. In *International Conference on Learning Representations*, volume 2026, pages 101537–101558, 2026.

Peiqi Liu, Zhanqiu Guo, Mohit Warke, Soumith Chintala, Chris Paxton, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Dynamem: Online dynamic spatio-semantic memory for open world mobile manipulation. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13346–13355. IEEE, 2025.

Max Sobol Mark, Jacky Liang, Maria Attarian, Chuyuan Fu, Debidatta Dwibedi, Dhruv Shah, and Aviral Kumar. Bpp: Long-context robot imitation learning by focusing on key history frames. *arXiv preprint arXiv:2602.15010*, 2026.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. *arXiv preprint arXiv:2406.02523*, 2024.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Jianing Qian, Qinhe Peng, Emmanuel Panov, Leonor Fermoselle, Dinesh Jayaraman, Bernadette Bucher, and Tarik Kelestemur. Expanding spatial and temporal context for robotic imitation learning with scene graphs. 2026.

Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 1134–1141. IEEE, 2018.

Rutav Shah, Rajat Kumar Jenamani, Xiaohan Zhang, Lingfeng Sun, Roberto Martín-Martín, Yuke Zhu, Deva Ramanan, and Karl Schmeckpeper. Scaling short-term memory of visuomotor policies for long-horizon tasks. *arXiv preprint arXiv:2606.16178*, 2026a.

Rutav Shah, Yisu Li, Femi Bello, Yuke Zhu, and Roberto Martín-Martín. Memory retrieval in visuomotor policies for long-horizon robot control. In *Proceedings of Robotics: Science and Systems*, 2026b.

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*, volume 2026, pages 18567–18602, 2026.

Oriane Siméoni, Huy V Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, et al. Dinov3. *arXiv preprint arXiv:2508.10104*, 2025.

Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. Scaling up memory for robotic control via experience retrieval. In *International Conference on Learning Representations*, volume 2026, pages 97142–97166, 2026.

TRI LBM Team, Jose Barreiros, Andrew Beaulieu, Aditya Bhat, Rick Cory, Eric Cousineau, Hongkai Dai, Ching-Hsin Fang, Kunimatsu Hashimoto, Muhammad Zubair Irshad, Masha Itkina, Naveen Kuppuswamy, Kuan-Hui Lee, Katherine Liu, Dale McConachie, Ian McMahon, Haruki Nishimura, Calder Phillips-Grafflin, Charles Richter, Paarth Shah, Krishnan Srinivasan, Blake Wulfe, Chen Xu, Mengchao Zhang, Alex Alspach, Maya Angeles, Kushal Arora, Vitor Campagnolo Guizilini, Alejandro Castro, Dian Chen, Ting-Sheng Chu, Sam Creasey, Sean Curtis, Richard Denitto, Emma Dixon, Eric Dusel, Matthew Ferreira, Aimee Goncalves, Grant Gould, Damrong Guoy, Swati Gupta, Xuchen Han, Kyle Hatch, Brendan Hathaway, Allison Henry, Hillel Hochsztein, Phoebe Horgan, Shun Iwase, Donovon Jackson, Siddharth Karamcheti, Sedrick Keh, Joseph Masterjohn, Jean Mercat, Patrick Miller, Paul Mitiguy, Tony Nguyen, Jeremy Nimmer, Yuki Noguchi, Reko Ong, Aykut Onol, Owen Pfannenstiehl, Richard Poyner, Leticia Priebe Mendes Rocha, Gordon Richardson, Christopher Rodriguez, Derick Seale, Michael Sherman, Mariah Smith-Jones, David Tago, Pavel Tokmakov, Matthew Tran, Basile Van Hoorick, Igor Vasiljevic, Sergey Zakharov, Mark Zolotas, Rares Ambrus, Kerri Fetzer-Borelli, Benjamin Burchfiel, Hadas Kress-Gazit, Siyuan Feng, Stacie Ford, and Russ Tedrake. A careful examination of large behavior models for multitask dexterous manipulation. 2025. URL https://arxiv.org/abs/2507.05331.

Marcel Torne, Andy Tang, Yuejiang Liu, and Chelsea Finn. Learning long-context diffusion policies via past-token prediction. *arXiv preprint arXiv:2505.09561*, 2025.

Marcel Torne, Karl Pertsch, Homer Walke, Kyle Vedder, Suraj Nair, Brian Ichter, Allen Z Ren, Haohuan Wang, Jiaming Tang, Kyle Stachowicz, et al. Mem: Multi-scale embodied memory for vision language action models. *arXiv preprint arXiv:2603.03596*, 2026.

Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

Limin Wang, Bingkun Huang, Zhiyu Zhao, Zhan Tong, Yinan He, Yi Wang, Yali Wang, and Yu Qiao. Videomae v2: Scaling video masked autoencoders with dual masking. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14549–14560. IEEE, 2023.

Ganlin Yang, Zhangzheng Tu, Yuqiang Yang, Sitong Mao, Junyi Dong, Tianxing Chen, Jiaqi Peng, Jing Xiong, Jiafei Cao, Jifeng Dai, et al. Eventvla: Event-driven visual evidence memory for long-horizon vision-language-action policies. *arXiv preprint arXiv:2606.20092*, 2026a.

Sizhe Yang, Juncheng Mu, Tianming Wei, Chenhao Lu, Xiaofan Li, Linning Xu, Zhengrong Xue, Zhecheng Yuan, Dahua Lin, Jiangmiao Pang, et al. Memorywam: Efficient world action modeling with persistent memory. *arXiv preprint arXiv:2606.20562*, 2026b.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *International Conference on Learning Representations*, volume 2025, pages 29687–29707, 2025.

Yihan Zeng, Minghao Ye, Yiyuan Chen, Yide Shentu, Philipp Wu, Zike Yan, and Zhongyu Li. Kemo: Event-driven keyframe memory for long-horizon robot manipulation with vla policies. *arXiv preprint arXiv:2606.23589*, 2026.

Shaopeng Zhai, Qi Zhang, Tianyi Zhang, Fuxian Huang, Haoran Zhang, Ming Zhou, Shengzhe Zhang, Litao Liu, Sixu Lin, and Jiangmiao Pang. A vision-language-action-critic model for robotic real-world reinforcement learning. *arXiv preprint arXiv:2509.15937*, 2025.

Junbo Zhang and Kaisheng Ma. Benchmarking long-horizon mobile manipulation in multi-room dynamic environments. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 83–89. IEEE, 2025.

Sizhe Zhao, Haozhe Xie, Weiyu Zhao, Chenchu Zhang, Huan Wang, Chenyang Wang, Qinglin Liu, and Shengping Zhang. Memory as Plans: world-action modeling with memory-grounded planning. *arXiv 2609.11561*, 2026.

Contents

1 Introduction **2**

2 Related Work **3**

3 Problem Formulation **4**

4 IMPRINT: Interaction Memory with Progress- and Reconstruction-Informed Training **5**
4.1 Multi-stream Memory Encoding 5
4.2 Decoupled Memory Supervision 6
4.2.1 Stage 1: Memory Pre-training 6
4.2.2 Stage 2: Policy-Agnostic Memory Adaptation 7

5 Experiments **7**
5.1 Experimental Setup 7
5.2 IMPRINT Outperforms Existing Memory-Augmented VLAs with Better Task Performance and more Efficient Inference 8
5.3 Richer Multi-Stream Inputs, Greater Capacity, and Recurrent Compression Enable More Effective Memory Encoding 9
5.4 Memory Pretraining and Joint Adaptation Improve Task Performance 10

6 Conclusion and Limitations **11**

7 Acknowledgments **11**

A Implementation Details **18**
A.1 IMPRINT Implementation 18
A.1.1 Multi-stream Input Processing and Tokenization 18
A.1.2 Memory Encoder Architectures 19
A.1.3 Memory Decoder and Pretraining Objectives 20
A.1.4 Integration with Different VLA Backbones 21
A.1.5 Training Hyperparameters and Resources 22
A.2 Baseline Implementations 22
A.3 Ablation Configurations 24

B Benchmarks and Evaluation Protocols **25**
B.1 RMBench 25
B.2 Customized RoboCasa Benchmark 26
B.3 Real-World Setup and Tasks 26
B.4 Demonstration Datasets 28
B.5 Rollout Protocol and Success Metrics 28
B.5.1 RMBench 28
B.5.2 RoboCasa 29
B.5.3 Real World 30
B.6 Statistical Reporting 31

C Detailed Experiment Results & Analysis For Main Paper **32**
C.1 Comparison with Memory-Augmented VLAs . . . 32
C.1.1 Per-task Results on RMBench . . . 32
C.1.2 Per-task Results on RoboCasa . . . 32
C.1.3 Detailed Real-world Results . . . 33
C.2 Memory Encoding . . . 34
C.2.1 Memory Capacity . . . 34
C.2.2 Recurrent and Full-attention Encoders . . . 36
C.3 Memory Learning and VLA Backbones . . . 36
C.3.1 Training Recipe and Stage 1 Objectives . . . 36
C.4 Analysis of the Learned Memory . . . 36
C.5 Inference-Time Analysis . . . 41

D Additional Ablation Results **42**
D.1 Ablations on Visual Encoder For Input Stream & Supervision Objective . . . 42
D.2 Ablation on Historical Input Streams . . . 43

E Per-Task Failure Analysis **44**
E.1 RMBench Analysis . . . 44
E.2 RoboCasa Analysis . . . 51
E.3 Real-World Analysis . . . 66

A Implementation Details

This section describes the implementation of IMPRINT, the baseline methods we compare to, and the configurations used in our ablations.

A.1 IMPRINT Implementation

IMPRINT is our proposed two-stage latent memory learning pipeline introduced in Sec. 4: multi-stream interaction history is first bootstrapped with reconstruction and task-progress estimation, and the resulting memory is then jointly adapted with a VLA for control. Following the notation in Sec. 3, B denotes the number of memory tokens and $m_t \in \mathbb{R}^{B\times d}$ denotes the memory at decision step t. The default description of IMPRINT uses the recurrent bottleneck; the full-attention bottleneck is a matched encoder variant trained with the same objectives and memory interface. The frozen video tokenizer performs within-stream compression, while the memory bottleneck integrates information across streams and over time.

A.1.1 Multi-stream Input Processing and Tokenization

Observations. For RMBench, each observation contains three synchronized RGB views in the fixed order head camera, left wrist camera, and right wrist camera. The simulator renders each view at 320×240 pixels. Before visual feature extraction, we resize each image to 256×256. We downsample the full history by 2 for memory construction. For customized RoboCasa, each observation contains three synchronized RGB views, ordered for memory construction as agentview left, agentview right, and wristview. Each view is rendered and stored at 256×256 pixels. For real world tasks, each observation contains three synchronized RGB views in the fixed order head camera, left wrist camera, and right wrist camera, corresponding to the head camera and the two wrist cameras. All cameras capture images at 640×480 pixels. We decode the recorded videos in RGB format and resize each image to 256×256 using bilinear interpolation with antialiasing before DINOv3 feature extraction.

Consolidated visual history. We encode finalized history chunks with the frozen `Wan2.2-5B` video VAE [Wan et al., 2025]. Its temporal factor is four retained frames; because memory retains every second simulator frame, one finalized chunk spans eight raw simulator frames. The VAE produces 48-dimensional latent features. We average-pool each camera's latent spatial grid to 2×2 tokens, yielding $3 \times 4 = 12$ tokens per finalized multiview chunk. A chunk enters consolidated memory only after all four retained frames are available.

Recent visual history. Frames following the last finalized video chunk (<= 3) are encoded by the frozen `dinov3` image encoder. At 256×256 resolution, ViT-B/16 produces a 16×16 patch grid after removing prefix tokens. We average-pool this grid to 8×8, producing 64 tokens of width 768 per view and 192 tokens per synchronized multiview frame. The same DINOv3 tensor is used as the recent visual input to the memory encoder and as the Stage–1 visual reconstruction target.

In some ablations, we used a separate SigLIP/PaliGemma representation from $\pi_{0.5}$ visual encoder and use for the reconstruction target from the VLM prefix, both with width 2048. We treat this as a distinct input profile.

Robot signals and causal alignment. RMBench uses a 14-dimensional absolute joint action/state representation, which is zero-padded to 16 dimensions for the memory model. The main configuration

uses raw action and raw proprioceptive vectors rather than first differences. For a sampled store variant with retained frame indices $f_0, \ldots, f_n$, the memory action stream is

$$a_0^{\text{mem}} = \mathbf{0}, \qquad a_j^{\text{mem}} = a_{f_{j-1}}^{\text{raw}} \quad (j > 0). \tag{A.1}$$

Thus, the observation at retained index f_j is paired only with an action that has already been executed. Importantly, this shift is constructed separately for every temporally sampled variant; indexing a canonical shifted sequence at skipped frame indices is not equivalent. The Stage–1 decoder target for query j remains the current, unshifted $a_{f_j}^{\text{raw}}$.

Language. The memory encoder is conditioned on one 2048-dimensional language embedding of the task instruction ℓ per episode from $\pi_{0.5}$ language encoder. Stage–1 and Stage–2 training use this stored vector.

A.1.2 Memory Encoder Architectures

Full-attention bottleneck. Separate projections map the consolidated video, recent visual, action, and proprioceptive streams to a shared memory width d_m. The two visual streams use linear layers, and actions and proprioception use two-layer MLPs. The projected tokens are concatenated along the token dimension. Modality is identified only through the separate projections. B learned queries, modulated by the task language embedding through FiLM [Perez et al., 2018], then pass through two pre-normalized Perceiver blocks [Jaegle et al., 2021]. Each block applies cross-attention from the queries to the history tokens with padding masked, self-attention among the queries, and a residual tanh-activated linear update. The output $m_t \in \mathbb{R}^{B \times d_m}$ is a fixed interface shared by all policy backbones. Table 1 lists the hyperparameters.

Table 1: Full-attention memory encoder hyperparameters.

Setting	Value
Memory width d	1024
Memory tokens B	256 (ablated over $\{1, 4, 16, 64, 256, 1024\}$)
Visual stream projections	Linear
Action / proprioception projections	2-layer MLP, GELU, hidden width 512
Language conditioning	FiLM on queries, 2048-d instruction embedding
Perceiver blocks	2, pre-LayerNorm
Attention heads	8 (cross- and self-attention)
Residual update	tanh-activated linear

Recurrent bottleneck. Our streaming encoder uses gated-delta recurrence Yang et al. [2025] to produce $B \times 1024$ memory tokens. Each finalized video chunk contributes four action–proprioception pairs followed by 12 pooled Wan tokens, forming a sequence of 20 tokens. Each token receives learned modality and within-chunk position embeddings, together with a projected normalized chunk index.

The encoder contains two gated-delta blocks, each with eight heads and an FP32 matrix state of size 128×128 per head. Each block applies pre-normalization, a width-four causal depthwise convolution, and SiLU before projecting queries, keys, values, write rates, and decay factors. At event index t, head

h updates its state as

$$S^h_{t,\text{ret}} = \delta^h_t S^h_{t-1}, \tag{A.2}$$

$$e^h_t = v^h_t - S^h_{t,\text{ret}} k^h_t, \tag{A.3}$$

$$S^h_t = S^h_{t,\text{ret}} + \beta^h_t e^h_t (k^h_t)^\top, \qquad y^h_t = S^h_t q^h_t. \tag{A.4}$$

Here, β^h_t is a sigmoid-bounded write rate and δ^h_t is a positive learned decay factor; their projection biases are initialized to -1 and -4, respectively. Padded events leave the state unchanged.

Language-conditioned learned queries read the recurrent states and combine the resulting features with recent DINOv3 tokens and robot signals through one full cross-attention/self-attention block. This bounded recent window is processed separately from the recurrent state. During inference, we cache the matrix states and the three preceding convolution inputs, update each finalized chunk once, and reset the caches between episodes.

A.1.3 Memory Decoder and Pretraining Objectives

Decoder architecture. During Stage 1, a Perceiver-IO decoder reconstructs historical observations and robot signals from latent memory m_t. We uniformly sample up to 16 distinct historical steps $s \le t$ without replacement, using all available steps for shorter prefixes. For each sampled step, we construct queries

$$q_{s,j} = e_{\text{step}}(s) + e_{\text{patch}}(j), \qquad q_{s,\text{sum}} = e_{\text{step}}(s) + e_{\text{sum}}, \tag{A.5}$$

where $e_{\text{step}}(s)$ is a learned embedding of historical step s, $e_{\text{patch}}(j)$ identifies a visual patch position across the three camera views, and e_{sum} is a learned summary embedding shared across steps. Each step has 192 patch queries that reconstruct DINOv3 features (64 patches per view) and one summary query that reconstructs its action and proprioceptive state. Sampled steps are sorted, and padded query groups are masked.

The decoder has two pre-normalized blocks with eight attention heads and feedforward width 8192. In each block, the 193 queries for a step first self-attend within that step, cross-attend to all memory tokens, and pass through a feedforward network. Queries from different steps do not self-attend to one another. Historical content reaches the decoder only through m_t; it receives neither target features nor a separate language input. Patch queries predict 768-dimensional DINOv3 features, while the summary query predicts 16-dimensional action and proprioceptive vectors. A separate learned global query attends to m_t to predict progress in $[0, 1]$.

Multi-stream reconstruction. Following Sec. 4.2.1, we supervise reconstruction of visual features, actions, and proprioceptive states from m_t. Let $\mathcal{K}_t$ contain the K valid historical steps sampled from the history prefix. The history-reconstruction component of Eq. 4.4 is implemented as

$$\mathrm{L}_{\text{rec}} = \frac{1}{K} \sum_{k \in \mathcal{K}_t} \Big[\frac{\lambda_v}{d_v} \big| \hat{z}^v_k - z^v_k \big|_2^2 \quad + \frac{\lambda_a}{d_a} \big| \hat{a}_k - a_k \big|_2^2 + \frac{\lambda_q}{d_q} \big| \hat{q}_k - q_k \big|_2^2 \Big], \tag{A.6}$$

where $\hat{z}^v_k, \hat{a}_k, \hat{q}_k$ denote decoder predictions for the sampled step k. The denominators count the elements in each target: $d_v = 192 \times 768$ for visual features across all patches and views, and $d_a = d_q = 16$ for actions and proprioceptive states. Before computing visual reconstruction error, we independently apply parameter-free LayerNorm to each predicted and target patch feature along its feature dimension. Visual targets remain frozen. Padded queries are excluded, and we use $(\lambda_v, \lambda_a, \lambda_q) = (1.0, 0.5, 0.5)$.

Progress estimation. During Stage 1 (Sec. 4.2.1), a learned global query q_{prog} cross-attends to all latent memory tokens m_t. The attention output is added to the query and passed through a learned linear head W_ρ followed by a sigmoid:

$$\hat{\rho}_t = \sigma\big(W_\rho\,[q_{\text{prog}} + \text{CrossAttn}\big(q_{\text{prog}}, m_t\big)]\big) \in (0,1). \tag{A.7}$$

The progress prediction is therefore computed only from the compressed history in m_t. We supervise it with the normalized timestep as a coarse approximation of task progress. For a demonstration with T steps indexed by $t = 0, \ldots, T-1$,

$$\rho_t = \frac{t}{T-1}, \qquad \mathcal{L}_{\text{prog}} = (\hat{\rho}_t - \rho_t)^2. \tag{A.8}$$

This target measures relative temporal position within the demonstration and requires no additional annotations. We weight this term by $\lambda_\rho = 0.5$ in Eq. 4.4. In preliminary experiments on the RoboCasa Wash and Return task, we also used reward labels from off-the-shelf reward models, RoboMeter and Robo-Dopamine, as progress targets. The normalized timestep led to higher success, likely because these labels were noisy on our demonstrations. We therefore use the normalized timestep as the progress target.

The Stage 1 objective is

$$\mathcal{L}_{\text{mem}} = \mathcal{L}_{\text{rec}} + \lambda_\rho \mathcal{L}_{\text{prog}}, \tag{A.9}$$

where $\mathcal{L}_{\text{rec}}$ is defined in Eq. A.6, with $(\lambda_v, \lambda_a, \lambda_q, \lambda_\rho) = (1.0, 0.5, 0.5, 0.5)$. The no-progress ablation sets $\lambda_\rho = 0$ and retains all input streams. Only the action and state tokenizer, memory encoder and decoder are trained in this stage; visual and language features are frozen or precomputed, and training requires no VLA policy. Stage 2 transfers the memory encoder and discards the decoder. The first three terms supervise fine-grained reconstruction at individual past steps across all three streams, while the last term provides coarse task-progress supervision. Similar to masked video modeling [Wang et al., 2023], reconstruction serves as a representation learning objective that prepares features for downstream adaptation. Our goal is to *prime* the memory bottleneck with multi-stream experience and task progress, without requiring perfect history reconstruction. This gives joint policy fine-tuning informative historical features to build on, helping reduce reliance on spurious memory shortcuts.

A.1.4 Integration with Different VLA Backbones

Memory adapter. Following Dai et al. [2026], we inject memory into the action expert rather than the VLM context. This choice outperforms VLM-context injection both in their study and in our RoboCasa experiments with heuristic memory (Fig. 4). Memory tokens are therefore never appended to the VLM prefix, and the VLM context length and key-value cache are unchanged. We insert an adapter into each memory-enabled action-expert layer, between the layer's native attention update and its feedforward network (Fig. 11). Let X_ℓ denote the action hidden states at layer ℓ. The adapter first reads memory through cross-attention, with action tokens as queries and the memory tokens $m_t \in \mathbb{R}^{B\times d}$ as keys and values. It then maps this read to feature-wise scale and shift parameters that modulate the normalized action features:

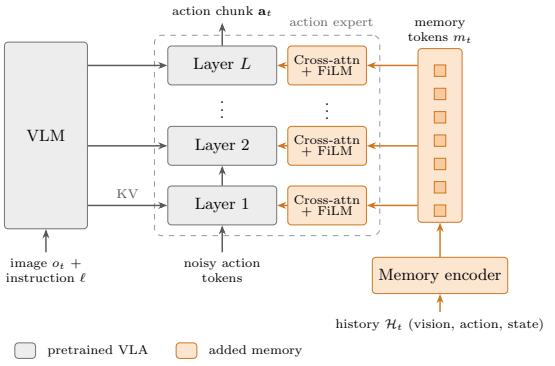


Figure 11: **Architecture of Memory Adapter for VLAs.**

$$R_\ell = \text{CrossAttn}\,\ell\big(\text{RMSNorm}(X\ell), \text{RMSNorm}(m_t)\big), \tag{A.10}$$

$$(\gamma_\ell, \beta_\ell) = W_\ell R_\ell, \qquad \widetilde{X}\ell = \text{RMSNorm}(X\ell) \odot (1+\gamma_\ell) + \beta_\ell. \tag{A.11}$$

The modulated features then pass through the original pre-feedforward normalization, feedforward network, and residual connection. Each layer has its own adapter parameters, while all layers read the same memory tokens. We initialize W_ℓ with small random weights and zero bias. This introduces only a small perturbation to the pretrained policy while allowing gradients to reach the memory attention and encoder from the start of training. Because the adapter attends to a fixed number of memory tokens, its cost is independent of history length. The adapter supplements rather than replaces each backbone's native conditioning, as detailed below.

π_0 and $\pi_{0.5}$. We add the adapter to every action-expert layer, with hyperparameters listed in Table 2. In π_0, the state token and the timestep embedding fused into each noisy-action token are unchanged. In $\pi_{0.5}$, the timestep-conditioned AdaRMSNorm remains active after memory modulation.

MolmoAct2. We inject memory only into the flow-matching action expert. Each memory-enabled layer applies action self-attention, VLM/state cross-attention, memory cross-attention and FiLM (Eqs. A.10–A.11), and the action feedforward network, in that order. MolmoAct2's native timestep-dependent scale, shift, and residual gates are applied after memory modulation, and its state embeddings remain in the native VLM/state key-value context.

Policy adaptation. Because all three backbones consume the same token interface, a single Stage 1 memory checkpoint initializes all of them, and only the action-expert adapter is backbone-specific. By default, we fine-tune memory and policy jointly with each backbone's native action objective, keeping the vision encoder frozen. For all models, we train the memory encoder, adapter, action expert, and VLM transformer. Table 6 summarizes alternative training configurations, and Sec. A.3 describes additional freezing choices.

Table 2: Memory adapter hyperparameters for π_0 and $\pi_{0.5}$.

Setting	Value
Memory / action-expert width	1024
Query heads	4
Key-value heads	1 (grouped-query attention)
Head dimension	256
Positional encoding	Separate rotary positions for memory and action tokens
FiLM projection initialization	$\mathcal{N}(0, 0.002^2)$, zero bias

A.1.5 Training Hyperparameters and Resources

Table 3 lists the RMBench optimization settings. Training uses four B200 or four H200 GPUs per run. Stage 2 uses each backbone's native flow-matching objective. For π_0 and $\pi_{0.5}$, we sample $\epsilon \sim \mathcal{N}(0, I)$ and $\tau \sim \text{Beta}(1.5, 1)$, form $x_\tau = \tau\epsilon + (1-\tau)a$, and regress the velocity $u_\tau = \epsilon - a$. MolmoAct2 uses the equivalent reverse-time convention. The standard recipe uses action supervision alone.

A.2 Baseline Implementations

We compare against three categories of memory-augmented policies, whose memory designs are summarized in Table 5. All six store only visual history, and none stores action or proprioceptive history.

Table 3: Optimization settings for IMPRINT on RMBench.

Setting	Stage 1: memory pretraining	Stage 2: joint adaptation
Updates	160,000	100,000
Global batch size	64	64
GPUs	4	4
Optimizer	AdamW	AdamW
Peak learning rate	1.0×10^{-4}	2.5×10^{-5}
Final learning rate	1.0×10^{-5}	2.5×10^{-6}
Warm-up updates	1,000	500
Schedule	cosine	cosine
Weight decay	1.0×10^{-4}	1.0×10^{-10}
Adam (β_1, β_2)	(0.9,0.999)	$(0.9, 0.95)$
Gradient clipping	1.0	1.0
EMA decay	–	0.999
Training seed	0	42
Checkpoint interval	10,000	10,000
Action horizon	–	50
Precision	mixed BF16/FP32	BF16

Heuristic methods retain raw frames chosen by a fixed rule. Uniform sampling [Dai et al., 2026] uses only visual history, in the form of frames sampled uniformly over the episode. EventVLA [Yang et al., 2026a] also uses only visual history: on RMBench, it concatenates the initial frame and a window of recent frames into the VLM input.

Hierarchical methods pair a high-level VLM with a low-level policy. MemER [Sridhar et al., 2026] uses visual history and text. Its high-level VLM takes the recent frames and previously selected keyframes as input to select new keyframes, and passes the next subtask to the low-level policy as text. Mem-0 [Chen et al., 2026b] likewise uses visual history and text. Its planner reads the initial frame and the text description and final frame of each completed subtask, and its executor attends to image features of the frame at the start of the current subtask and of recent frames.

Latent methods learn continuous memory representations. HAMLET [Koo et al., 2026] uses only visual history, encoding each past observation into moment tokens initialized by time-contrastive learning and aggregating them over a fixed window of past steps. MemoryVLA [Shi et al., 2026] also uses only visual history, storing perceptual tokens and a VLM cognitive token for each observation in a memory bank and retrieving from it at each step.

None of these methods uses action or proprioceptive history, both of which IMPRINT encodes alongside visual history.

Source of baseline results. Table 4 lists the backbone of each method and the source of its RMBench result. For Mem-0, MemER, EventVLA, and MemoryVLA, we report the RMBench success rates published in prior work, which follow the same protocol of 50 demonstrations for training and 100 evaluation rollouts per task. The Mem-0 result is reported by Chen et al. [2026b], and the MemER, EventVLA, and MemoryVLA results by Yang et al. [2026a]. For EventVLA, this is the visual-anchor configuration that its authors evaluate on RMBench, which retains only the initial frame and a few recent frames without its keyframe evidence memory, so it does not reflect the full EventVLA. For MemoryVLA, it is the QwenOFT variant. We train the non-memory VLAs, uniform sampling, and HAMLET ourselves on RMBench, and train all baselines ourselves on RoboCasa and in the real world. Because the reported results come from different backbones and training pipelines, they are not controlled comparisons. Controlled comparisons are provided by the baselines we train and by the ablations in Sec. 5.3 and

Table 4: Compared policies, their backbones, and the source of their RMBench results.

Method	Category	Backbone	RMBench result
non-memory VLA	None	π_0, $\pi_{0.5}$, MolmoAct2	Trained by us
Uniform sampling	Heuristic	$\pi_{0.5}$	Trained by us
EventVLA (visual anchors only)†	Heuristic	QwenOFT (Qwen3-VL-4B)	Yang et al. [2026a]
MemER	Hierarchical	VLM planner + VLA policy	Yang et al. [2026a]
Mem-0	Hierarchical	Qwen3-VL-8B planner + DiT policy	Chen et al. [2026b]
HAMLET	Latent	GR00T N1.6	Trained by us
MemoryVLA (QwenOFT)†	Latent	QwenOFT	Yang et al. [2026a]
IMPRINT	Latent	π_0, $\pi_{0.5}$, MolmoAct2	Trained by us

Sec. 5.4.

A.3 Ablation Configurations

Table 6 defines every variant used in our ablations and indicates where each is reported. The main paper studies four aspects of IMPRINT: memory encoding (Sec. 5.3), the training recipe and pretraining objectives (Sec. 5.4), the choice of VLA backbone, and where memory is injected into the policy. Each ablation changes a single factor relative to the default configuration described in Sec. A.1. Unless stated otherwise, the demonstrations, tokenizers, action horizon, optimizer, training budget, and evaluation protocol are held fixed, and all variants are evaluated with the same number of trials per task as the main results.

Memory encoding. The capacity ablation (Fig. 7) varies only the number of memory tokens, $B \in \{1, 4, 16, 64, 256, 1024\}$, while holding the memory width, tokenizers, encoder depth, pretraining objectives, and policy adapter fixed. It uses the full-attention encoder and is evaluated on RMBench and on the memory and non-memory task subsets of RoboCasa. The encoder ablation (Sec. 5.3) keeps $B = 256$ and replaces the recurrent gated-delta encoder with the full-attention Perceiver encoder described in Sec. A.1.2. Both encoders are evaluated on RMBench and RoboCasa, and their latency is measured with the protocol in Sec. C.5.

Training recipe. The recipe ablation (Fig. 8, left) compares three ways of training the memory encoder. *End-to-end* initializes the encoder randomly and trains it only through the action loss in Eq. 3.3. *Frozen memory* initializes the encoder from Stage 1 and keeps it fixed during Stage 2, so only the adapter and policy are trained. IMPRINT initializes the encoder from Stage 1 and updates it jointly with the adapter and policy. All three variants are evaluated on RMBench, and the same checkpoints are used for the representation analysis in Sec. C.4.

Pretraining objectives. The objective ablation (Fig. 8, middle) varies the Stage 1 loss weights in Eq. 4.4 while keeping Stage 2 adaptation, the decoder, and the sampling of up to 16 historical steps fixed. *Without progress* sets $\lambda_\rho = 0$. *Visual only* keeps only visual reconstruction, and *action and state only* keeps only action and proprioceptive reconstruction. These variants remove supervision but not inputs, so every variant still encodes all three streams. All variants are evaluated on RMBench.

Backbones and memory injection. The backbone study (Fig. 10) applies the same memory pipeline and adaptation procedure to π_0, $\pi_{0.5}$, and MolmoAct2, changing only the backbone-specific adapter described in Sec. A.1.4. Each backbone is compared with its non-memory counterpart trained with the

Table 5: Comparison of memory designs. ✓: yes; ~: partially; ✗: no. †On RMBench, EventVLA is evaluated only in its visual-anchor configuration, which keeps the initial frame and a few recent frames and omits its keyframe evidence memory [Yang et al., 2026a]. The reported result therefore reflects this heuristic variant rather than the full EventVLA, and the column describes this evaluated variant.

	Heuristic		Hierarchical		Latent		
	Uniform sampling	EventVLA†	MemER	Mem-0	HAMLET	MemoryVLA	**IMPRINT**
What memory stores							
Visual history	✓	✓	✓	✓	✓	✓	✓
Action history	✗	✗	✗	✗	✗	✗	✓
Proprio. history	✗	✗	✗	✗	✗	✗	✓
Memory form	Frames	Frames	Frames, text	Frames, text	Latent	Latent	Latent
How memory is supervised							
Action loss	✓	✓	✗	✓	✓	✓	✓
Visual objective	✗	✗	✗	✗	✓	✗	✓
Action/proprio. objective	✗	✗	✗	✗	✗	✗	✓
Subtask/keyframe labels	✗	✗	✓	✓	✗	✗	✗
Task progress	✗	✗	✗	✗	✗	✗	✓
Training and inference							
No annotations needed	✓	✓	✗	✗	✓	✓	✓
Trained without the policy	✗	✗	✓	✓	✓	✗	✓
No high-level VLM	✓	✓	✗	✗	✓	✓	✓
Covers the full history	✗	✗	✗	✗	✗	✓	✓
Shown on multiple VLAs	✗	✗	✗	✗	✓	✗	✓

same Stage 2 budget on RMBench. The injection study (Fig. 4) supplies the same uniformly sampled frames either to the action expert through the memory adapter or to the VLM context as additional image tokens, and is evaluated on RoboCasa.

B Benchmarks and Evaluation Protocols

We evaluate memory-based manipulation tasks on RMBench, memory and non-memory tasks on a customized RoboCasa benchmark, and three real-world tasks. The following sections describe the tasks, datasets, and evaluation procedures.

B.1 RMBench

RMBench comprises nine manipulation tasks built on RoboTwin 2.0 with the Aloha-AgileX embodiment. It distinguishes tasks requiring recall of one event, M(1), from those requiring a sequence of events, M(n). We use the demo clean setting, original task instructions, and 50 expert demonstrations per task, for 450 demonstrations in total. Table 7 summarizes the tasks and episode limits.

Episode seeds determine object identities, poses, colors, and hidden task variables. Success follows each environment's state-based criteria. For example, Cover Blocks requires the complete cover/uncover sequence and rejects out-of-order transitions.

Table 6: Configurations used in the ablation study. *Stage 1 init.*: whether the memory encoder starts from Stage 1 pretraining. *Memory updated*: whether the encoder is optimized during Stage 2. Unlisted settings match the default IMPRINT configuration.

Study	Variant	Stage 1 init.	Memory updated	Change from default	Reported in
Encoding	Memory capacity	yes	yes	$B \in \{1, 4, 16, 64, 256, 1024\}$ (full-attention encoder)	Fig. 7
	Full attention	yes	yes	Perceiver attention instead of recurrence	Fig. 6, Sec. 5.3
Recipe	End-to-end	no	yes	Random encoder, action loss only	Fig. 8
	Frozen memory	yes	no	Encoder fixed during Stage 2	Fig. 8
	IMPRINT	yes	yes	None (default)	Fig. 8
Objectives	Without progress	yes	yes	$\lambda_\rho = 0$	Fig. 8
	Visual only	yes	yes	$\lambda_a = \lambda_q = \lambda_\rho = 0$	Fig. 8
	Action and state only	yes	yes	$\lambda_v = \lambda_\rho = 0$	Fig. 8
Integration	Backbone	yes	yes	π_0, $\pi_{0.5}$, or MolmoAct2	Fig. 10
	Injection location	–	–	Sampled frames in action expert or VLM context	Fig. 4
Inputs	Vision only	yes	yes	Action and state inputs and targets removed	Table 20
	Vision + Action	yes	yes	State input and target removed	Table 20
	Vision + State	yes	yes	Action input and target removed	Table 20

B.2 Customized RoboCasa Benchmark

Our customized RoboCasa benchmark contains 12 tasks, six memory-based and six non-memory (Table 8). Four of the memory tasks follow ReMemBench [Shah et al., 2026a] and require recalling an object's location or its association with a container: Retrieve Fruit, Retrieve Oil, Wash and Return, and Wash and Return to Same Location. The other two, Sweeten Hot Chocolate and Rinse Cutting Board, require tracking task progress. In each memory task, the policy must retain a variable that the current observation no longer reveals, such as an earlier object location, a source container, an exact initial position, the number of sugar cubes already deposited, or the accumulated rinse duration. The non-memory tasks can be specified from the current observation and instruction alone, although they may still require multi-step closed-loop manipulation.

Each task has 50 expert demonstrations, for 600 demonstrations in total. For evaluation, we restore 50 recorded initial states per task, each consisting of the simulator XML, state, and episode metadata, rather than sampling new scenes procedurally. These 50 configurations do not correspond to 50 distinct kitchens. Some tasks use a single layout and style and vary objects, placements, and task parameters, whereas others span up to 34 layouts and 35 styles. Policies observe three synchronized 256×256 RGB views as left agentview, right agentview, and wrist view. An episode ends upon success or after a task-specific horizon of policy steps. Retrieve Fruit and Retrieve Oil begin with a scripted exploration phase where it moves around left to right to view the full scene before the policy takes control, so their total simulator duration can exceed the listed horizon.

B.3 Real-World Setup and Tasks

We use a bimanual YAM robot with one head camera and two wrist cameras. The robot is controlled in joint space with 14-dimensional action and state vectors, padded to 32 dimensions for the π_0 inter-

Table 7: RMBench tasks and episode limits, measured in simulator steps.

Task	Type	Task and memory requirement	Limit
Observe and pickup	M(1)	Pick the object matching a target shown before occlusion.	250
Rearrange blocks	M(1)	Move the center block to the empty mat, press, and move the initially mat-placed block to the center.	700
Put back block	M(1)	Move the block to the center, press, and return it to its original mat.	500
Swap blocks	M(1)	Use the spare tray to exchange the blocks' initial tray assignments, then press once.	1000
Swap T	M(1)	Exchange the initial positions and orientations of two T-shaped blocks.	600
Battery	M(n)	Use gauge feedback from tested orientation pairs to place both batteries correctly.	1000
Block ranking	M(n)	Use feedback from failed arrangements to find the hidden left-to-right block order.	3500
Cover blocks	M(n)	Remember the initial color order, cover from left to right, and uncover in red–green–blue order.	1500
Press button	M(n)	Recall two displayed numbers, press the left and middle buttons those numbers of times, and confirm.	1500

face. Real-world policies predict chunks of 30 actions and execute all of them for all methods. The visualization of those tasks are shown in Fig. 2 and Fig. 12.

Dust Table. The task instruction is "move the objects away to dust the table and put them back in the original positions." The robot must remember the objects' original locations while clearing and cleaning the table. It must also remember whether the table has already been dusted, to avoid dusting it repeatedly. The original locations are recoverable from visual history, while the dusting progress is reflected in the robot's past actions and proprioceptive states as well as visual histories. The objects are placed on the table in two sets of bottles. For each set, the left and right positions of the bottles are swapped with probability 0.5, and the duster is placed on the left or right side of the table with equal probability.

Scan Bottle. The instruction is "scan the bottles and put them on the other side." The robot scans and transfers the bottles while tracking which have already been scanned. The bottle locations are recoverable from visual history, while the scanning status and how long it has held still is reflected in the robot's past actions and proprioceptive states. The bottles start on either the left or the right side of the table, and we randomize which arm performs the scan and which bottle is scanned first.

Fold Cloth. The instruction is "follow the example to fold the towel." The robot must retain the demonstrated folding configuration and the motion to fold the cloth while executing the fold. The towels come in four colors, and each can be folded in one of four styles as shown in Fig. 13. For each episode, one robot demonstration with the same color and style is provided as an in-context example and concatenated to the memory. Imitating the example requires its actions and proprioceptive states to reproduce the folding motion.



Figure 13: **Real World Fold Cloth Task Configuration**

Table 8: Tasks in the customized RoboCasa benchmark. *Horizon*: maximum number of policy-controlled simulator steps. *L/S*: number of distinct layouts and styles among the 50 evaluation configurations. Instructions are abbreviated.

Task	Instruction and memory requirement	Horizon	L/S	Variation across configurations
Memory				
Retrieve fruit	Place the {fruit} in the sink; recall which side of the sink it was observed on.	1600	1/1	9 fruit categories; target at the far left or right of the sink.
Retrieve oil	Pick up the olive-oil bottle; recall which of two bottles is the target after both sides are shown.	1600	1/1	Four initial/target-side combinations (LL, LR, RL, RR).
Wash and return	Wash the fruit and return it to its original container; recall the source container.	1600	1/1	Left or right source container; fruit, receptacle, and placements vary.
Wash and return to same location	Wash the fruit and return it to its exact initial location; recall that location.	1600	1/1	Initial fruit location varies.
Sweeten hot chocolate	Place {1, 2, 3} sugar cubes in the saucepan; track how many have been placed.	2100	15/17	Target counts 1/2/3 in 20/20/10 configurations; 3–5 cubes per scene, including 0–3 extras.
Rinse cutting board	Rinse the board with hot water for ten seconds, then turn off the faucet; track rinse duration.	2250	34/18	Board instance, pose, sink, and kitchen vary.
Non-memory				
Pan transfer	Dump the vegetables from the pan onto the plate and return the pan to the stove.	1800	1/1	Pan, food, plate, and placements vary.
Organize condiments	Move only the condiments or shakers from the counter into the cabinet.	2250	33/35	Three targets and one distractor; 8 target and 36 distractor categories.
Counter to sink	Pick the {object} from the counter and place it in the sink.	600	28/29	29 target categories, 43 target assets, and two distractors per scene.
Place equal ice cubes	Place two of the four ice cubes into each glass of water.	2850	25/34	Four cubes, two glasses, and one bowl, drawn from 24 assets.
Scrub cutting board	Scrub the board with the sponge, then release it.	1200	30/32	Six cutting-board and four sponge assets.
Frying-pan adjustment	Move the pan to another burner and turn that burner on.	1200	31/30	Ten pan assets; source and destination burners vary.

B.4 Demonstration Datasets

We use 50 demonstrations per task for 9 tasks in Rmbench, 50 demonstrations per task for 12 Robocasa tasks and 100 demonstrations per task for 3 real world tasks. Rmbench demonstrations are collected with scripted policies. Robocasa uses human teleoperated demos with spacemouse from Robocasa and Remembench. Real world task uses leader-follower arms in bimanual Yam platforma to collect demonstrations with kinesthetic teaching. We randomize the object locations and the memory condition for each demonstration in the real world tasks. The representation analyses in Sec. C.4 use training demonstrations to characterize the learned memory.

B.5 Rollout Protocol and Success Metrics

We evaluate policies on two simulation benchmarks, RMBench and RoboCasa, and on three real-world tasks. For each setting, we first describe the rollout protocol and then the task success criteria.

B.5.1 RMBench

We follow the evaluation protocol of [Chen et al., 2026b], running 100 accepted trials per task starting from seed 100000. Each task runs on two L40S GPUs: one serves the policy, and the other handles simulation and rendering. The policy deterministically predicts and executes chunks of 50 actions, using a causal memory buffer that holds finalized history chunks and a recent observation window. Success criteria for all 9 tasks follow [Chen et al., 2026b].



Figure 12: **Real World Tasks with Three Camera Views**

B.5.2 RoboCasa

Protocol. We run 50 rollouts for each of the 12 tasks, one per fixed, demonstration-derived scene configuration. Each rollout restores the corresponding simulator state, object placements, instruction, and camera configuration. The $\pi_{0.5}$ policy deterministically predicts 20-action chunks and executes five actions before replanning. All tasks use the same three-view observation pipeline as in training, consolidate finalized history chunks, and retain the recent-observation window.

Tasks differ in how the episode is initialized. For retrieval tasks, we reconstruct the recorded exploration prefix by replaying simulator states and inserting the associated 12-dimensional actions into memory with the same causal alignment used in training; these actions populate memory but are not re-executed. Wash-and-return tasks have no scripted exploration and start directly from the restored reset state, and all remaining tasks start from their demonstration-aligned reset.

Success criteria. Success is determined by task-specific predicates on the simulator state. Throughout, we say an object is *released* when the gripper finger gap exceeds 0.075 and the end-effector–object distance exceeds 0.10 m.

Memory tasks.

- **Retrieve Fruit.** The policy must remember whether the instructed fruit is on the left or right and place it in the sink. Success requires the target fruit to be fully inside the sink basin and released.
- **Retrieve Oil.** The policy must remember which oil bottle is the target and retrieve it from the counter. Success requires the target bottle to be lifted more than 0.08 m above its initial height.
- **Wash and Return Left/Right.** The policy must place the fruit in the sink and then return it to its remembered source container. The evaluator first requires a full sink-entry event (with 0.03 m of vertical padding on the sink region). Afterwards, the fruit's x and y coordinates must each lie within 0.08 m of the source-container center, and the fruit must be released.

- **Wash and Return to the Same Location.** Same as above, except the fruit must be returned to its original counter position: after sink entry, $|x - x_0| < 0.05$ m and $|y - y_0| < 0.05$ m, and the fruit must be released. Success is latched once all conditions are met.

- **Sweeten Hot Chocolate.** The policy must place the instructed number $g \in \{1, 2, 3\}$ of sugar cubes into the saucepan and then return the arm toward its reset pose. Any cube, including distractors, counts toward g. A placement event is recorded when a cube stays inside the saucepan, ungrasped, for 10 consecutive frames. Success is computed from the stored trajectory and requires exactly g placement events, no cube grasped at the end, maximum mobile-base joint drift of at most 2×10^{-4}, and at least one post-placement end-effector position within 0.15 m of its initial position.

- **Rinse Cutting Board.** The policy must hold the cutting board under running water and then close the faucet. The rinse event is latched once the board stays under water, with the faucet on and water temperature at least 0.6, for more than 100 consecutive evaluator steps. Afterwards, the faucet must be off, with its handle within 0.2 rad of the closed position.

Non-memory tasks.

- **Pan Transfer.** The policy must move the vegetable from the pan to the plate while leaving the pan on its original burner. Success requires the vegetable to be inside the plate, and the pan to be released, in contact with the stove, centered within 0.08 m of the recorded burner, and tilted less than 15° from upright. Any gripper–vegetable contact during the rollout invalidates success.

- **Organize Condiments.** The policy must move three condiments into the cabinet while leaving a distractor on the counter. Success requires all three targets inside the cabinet, the distractor in contact with the counter, and the end effector more than 0.25 m from all four objects. The cabinet-door state is not checked.

- **Pick and Place from Counter to Sink.** Success requires the target object's center to lie inside the sink and the end effector to be more than 0.25 m from the object.

- **Place Equal Ice Cubes.** The policy must distribute four ice cubes so that each of two glasses contains exactly two. A cube counts as inside a glass if it is detected through physical contact within the native 0.5 m receptacle threshold, or if it touches a cube detected this way; each glass must contain at least one directly detected cube. At completion, the end effector must be more than 0.15 m from every cube.

- **Scrub Cutting Board.** The policy must grasp the sponge, scrub the board over a sufficiently large area, and release the sponge. While the grasped sponge contacts the board, the evaluator must record at least five distinct contact positions, each at least 0.02 m from all earlier ones, spanning a planar range of at least 0.10 m. The final end-effector–sponge distance must exceed 0.15 m.

- **Frying-Pan Adjustment.** The policy must move the pan to a different burner and turn that burner on. Success requires the pan to contact the stove with its center within 0.08 m of a burner other than the starting one, and the destination burner's knob to be on. No release condition is imposed.

B.5.3 Real World

Protocol. Table 9 summarizes the rollouts for each task. Before every rollout, the operator manually resets the workspace and randomizes object placement. The π_0 policy predicts 30-step, 14-dimensional joint-and-gripper action chunks. Real-time chunking is disabled, and each full chunk is executed synchronously at a nominal 30 Hz before replanning, corresponding to roughly one second of motion plus

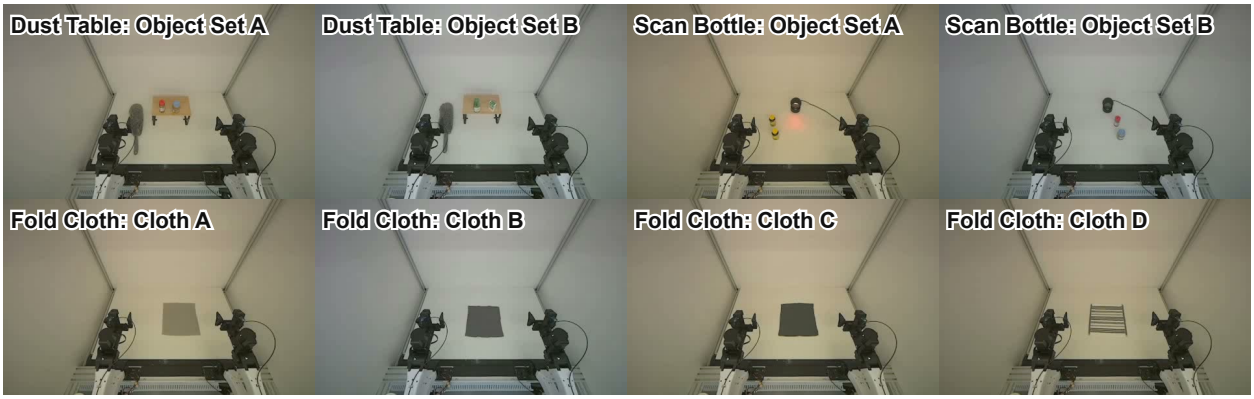


Figure 14: **OOD Scenarios for Each Task in the Real World**

policy-refresh latency. For Dust Table and Scan Bottle, episodic memory starts empty and is populated online with the observations, robot states, and actions actually issued to the robot. For Fold Cloth, memory is initialized with a full demonstration as an in-context example, which is encoded as context but not physically replayed.

Table 9: Real-world rollout protocol.

Task	Rollouts	Variation
Dust Table	20	2 object sets × duster left/right (5 rollouts each)
Scan Bottle	20	bottles on left/right (10 rollouts each)
Fold Cloth	32	16 towel color/style contexts (2 rollouts each)

Success criteria.

- **Dust Table.** The robot must place the objects down upright, dust the table exactly once, and return the objects upright to the small table in the correct left/right arrangement.
- **Scan Bottle.** The robot must pick up the bottles one at a time, scan each exactly once, and place them on the other side in an aligned vertical row without collisions.
- **Fold Cloth.** The robot must reproduce the folding steps of the demonstration and reach a final shape close to the demonstrated one. Repeated grasp attempts are allowed.

OOD Evaluation. To test whether the memory overfits to the objects and scenes seen in the demonstrations, we also evaluate on out-of-distribution objects. A good memory objective should extract the information needed for the task while still generalizing to new objects. For Dust Table, we introduce two new sets of objects whose shapes differ from the in-distribution objects. For Scan Bottle, we likewise introduce two new sets of objects. For Fold Cloth, we use four new towels in colors not seen during training (Fig. 14). All OOD evaluations follow the same protocol and number of rollouts as the in-distribution evaluation.

B.6 Statistical Reporting

We follow the statistical reporting protocol of [Team et al., 2025]. Each rollout is treated as an independent Bernoulli trial, so estimating a policy's success rate on a task amounts to estimating the parameter

p of the Bernoulli distribution that generates its success/failure outcomes. In simulation, initial conditions are set by fixed random seeds; in the real world, the operator resets and randomizes the scene before every rollout.

Violin plots. Each violin shows the Bayesian posterior over p given the observed outcomes, under a uniform prior. With s successes out of n rollouts and a Beta$(1,1)$ prior, the posterior is Beta$(s{+}1,\ n{-}s{+}1)$. The width of the violin at success rate x is proportional to the posterior density at x, and the horizontal marker denotes the empirical success rate s/n. The violin thus reflects uncertainty in the true success rate given the number of rollouts, not variation across tasks.

Aggregation across tasks. For aggregate plots, we pool outcomes across the tasks of a benchmark and plot Beta$\big(\sum_i s_i + 1,\ \sum_i (n_i - s_i) + 1\big)$. Because all tasks within a benchmark use the same number of rollouts, the pooled success rate equals the unweighted mean of per-task success rates. As noted by Team et al. [2025], pooling treats outcomes from different tasks as exchangeable rather than strictly i.i.d.

C Detailed Experiment Results & Analysis For Main Paper

This section reports per-task results for every experiment in Sec. 5 and follows the same order. Sec. C.1 corresponds to the comparison with existing memory-augmented VLAs (Sec. 5.2), Sec. C.2 to memory encoding (Sec. 5.3), and Sec. C.3 as well as Sec. C.4 to memory learning and VLA backbones (Sec. 5.4). Unless noted otherwise, all entries are task success rates in percent. RMBench uses 100 evaluation rollouts per task and RoboCasa uses 50. Averages are task-balanced: we first compute success within each task and then average tasks with equal weight.

C.1 Comparison with Memory-Augmented VLAs

C.1.1 Per-task Results on RMBench

Table 10 groups the RMBench tasks by whether they require recalling a single event, M(1), or a sequence of events, M(n) (App. B.1). On M(1) tasks, several methods perform well once the relevant frame is retained. EventVLA, which always keeps the initial frame, averages 79.0% on these tasks, close to 81.0% for recurrent IMPRINT. The gap widens on M(n) tasks, where recurrent IMPRINT averages 62.5%, compared with 54.0% for EventVLA and at most 38.8% for the other baselines. Press Button, which requires counting how many times each button has been pressed, shows this difference most clearly. Every baseline stays at or below 3%, whereas IMPRINT reaches 21% with full attention and 31% with the recurrent encoder. Because each press doesn't change the scene and the robot needs to keep along history to count the press properly, only IMPRINT retains the history with the proper compression. Observe and Pickup and Battery remain difficult for all methods, with no method exceeding 21% and 41%, respectively because the former has a diverse set of similar objects that might be hard to tell and the latter requires high precision to grasp the round objects as battery and failure of grasping is irrecoverable.

C.1.2 Per-task Results on RoboCasa

Table 11 shows that naively adding historical frames to the policy does not reliably help. Both heuristic variants have lower overall success than the non-memory $\pi_{0.5}$ (48.8% and 41.5% vs. 52.2%). Supplying the frames as additional VLM-context images gives almost no benefit on memory tasks (39.0%

Table 10: Per-task success rates on the nine RMBench memory tasks, grouped into single-event, M(1), and multi-event, M(n), tasks. The final column is the task-balanced average reported in Fig. 3. The best result for each task and for the average is shown in bold. †Results reported by prior work (App. A.2).

	M(1)					M(n)				
Method	Observe / pick	Rearrange blocks	Put back block	Swap blocks	Swap T	Battery	Block ranking	Cover blocks	Press button	Average
$\pi_{0.5}$ (no memory)	8	63	20	16	8	30	66	15	3	25.4
Uniform sampling	5	79	**100**	62	30	28	82	13	3	44.7
EventVLA†	**21**	96	95	96	87	35	81	97	3	67.8
MemER†	7	17	0	14	7	27	0	6	0	8.7
Mem-0†	4	89	90	67	14	28	18	68	0	42.0
HAMLET	6	76	23	25	13	26	70	14	0	28.1
MemoryVLA†	2	53	81	76	9	33	53	69	0	41.7
IMPRINT (full)	9	**100**	**100**	**100**	**100**	**41**	83	**100**	21	72.7
IMPRINT (recurrent)	6	99	**100**	**100**	**100**	35	**84**	**100**	**31**	**72.8**

vs. 38.7%). On Retrieve Oil and Wash and Return, for example, it performs within 2 points of the non-memory policy (50% vs. 48% and 46% vs. 46%). Supplying the same frames to the action expert is more effective, raising memory-task success to 54.0%. The largest gains appear when the required memory is a single visual snapshot: 88% on Retrieve Fruit and 98% on Retrieve Oil, where the policy only needs to recall where the target was seen. We attribute this difference to the injection point. Adding images to the VLM context changes the input distribution of the pretrained VLM, whereas injecting memory into the action expert leaves the VLM input unchanged, consistent with the findings of Dai et al. [2026]. IMPRINT therefore uses the action expert as its injection point.

Neither heuristic variant, however, preserves performance on non-memory tasks. Adding raw frames lowers success from 65.7% to 43.7% in the action expert and to 44.0% in the VLM context for non-memory tasks. Examples include Pan Transfer (74% to 48% and 44%), Organize Condiments (72% to 32% and 48%), and Frying-Pan Adjustment (76% to 52% and 22%). When the current observation is sufficient, the extra frames likely act as distractors that the policy cannot learn to ignore.

IMPRINT injects memory at the same point as the action-expert variant, so the remaining gap reflects the memory representation rather than the injection point. Replacing raw frames with compressed multi-stream memory raises success from 54.0% to 65.3% for memory tasks and 43.7% to 68.0% for non-memory tasks, above the non-memory $\pi_{0.5}$. The gains come mainly from tasks that depend on what the robot has already done. Recurrent IMPRINT improves Wash and Return from 58% to 96%, Wash and Return to Same Location from 36% to 58%, and Sweeten Hot Chocolate from 36% to 64%. On non-memory tasks, it matches or exceeds the non-memory $\pi_{0.5}$ on four of six tasks. Rinse Cutting Board stays at or below 8% and Place Equal Ice Cubes at or below 20% for every method.

C.1.3 Detailed Real-world Results

Table 12 reports the rollout counts behind Fig. 5. IMPRINT improves over the strongest baseline, uniform sampling, by 30, 25, and 25 points on Dust Table, Scan Bottle, and Fold Cloth.

The baselines fail in different ways. MemER is limited by its hierarchical design. In our rollouts, its high-level VLM often predicts the wrong subtask, for example returning to a bottle that has already been scanned. Even when the prediction is correct, the low-level policy's actions change abruptly when the subtask switches, which frequently disrupts an ongoing grasp or placement. HAMLET does not improve over π_0 on any task. Its memory often retains the wrong information, possibly because the

Table 11: Per-task success rates on the customized RoboCasa benchmark (Fig. 4). *Heuristic action* supplies uniformly sampled historical frames to the action expert, whereas *heuristic VLM* supplies them as additional VLM-context images. The best result for each task and each average is shown in bold.

Task	Type	$\pi_{0.5}$	Heuristic action	Heuristic VLM	IMPRINT full	IMPRINT recurrent
Retrieve fruit	Memory	42	88	60	84	**90**
Retrieve oil	Memory	48	**98**	50	76	80
Wash and return	Memory	46	58	46	86	**96**
Wash and return to same location	Memory	46	36	44	**62**	58
Rinse cutting board	Memory	4	**8**	0	0	4
Sweeten hot chocolate	Memory	46	36	34	58	**64**
Pan transfer	Non-memory	74	48	44	**76**	72
Frying-pan adjustment	Non-memory	**76**	52	22	72	70
Organize condiments	Non-memory	72	32	48	62	**74**
Place equal ice cubes	Non-memory	16	6	6	6	**20**
Scrub cutting board	Non-memory	60	50	56	66	**76**
Counter to sink	Non-memory	96	74	88	**98**	96
Memory task average	–	38.7	54.0	39.0	61.0	**65.3**
Non-memory task average	–	65.7	43.7	44.0	63.3	**68.0**
Overall average	–	52.2	48.8	41.5	62.2	**66.7**

time-contrastive objective is not learning the critical information for compressing the memory. Its action predictions are also imprecise and produce jerky motions that cause manipulation failures even when the remembered information is correct.

Fold Cloth is the lowest-scoring task for π_0, HAMLET, and uniform sampling. Reproducing the in-context example requires both its visual appearance, to identify the folding style, and the demonstrated actions and proprioceptive states, to reproduce the folding motion. Baselines that retain only visual history can observe the example but not recover how it was executed. IMPRINT encodes all three streams of the example and reaches 72% on this task.

Under the OOD object variations (App. B.5.3), IMPRINT retains most of its performance on Scan Bottle (70% vs. 75%) but drops on Dust Table (45%) and Fold Cloth (31%), which involve longer horizons and more precise placement.

C.2 Memory Encoding

C.2.1 Memory Capacity

Table 14 shows that task performance generally improves with memory capacity up to $B = 256$. As the number of memory tokens grows from 4 to 256, average success rises on RMBench (67.0% to 72.7%) and on RoboCasa memory tasks (54.0% to 61.0%). Non-memory success stays near 57% through 64 tokens and rises to 63.3% at 256. Additional capacity mainly helps tasks that require retaining several pieces of information. On RMBench, Cover Blocks, which requires the order of three colors, rises from 68% with 4 tokens to 100% with 256 tokens, and Block Ranking rises from 73% to 83%. On RoboCasa, Sweeten Hot Chocolate improves steadily from 40% to 58%. In contrast, tasks that depend on a single location or assignment, such as Rearrange Blocks and Put Back Block, already reach 100% with a single

Table 12: Real-world results (Fig. 5). Each cell reports successful rollouts / total rollouts, followed by the success rate in parentheses. The average is the mean of the per-task success rates. The best in-distribution result for each task and the average is shown in bold. OOD denotes the held-out object variations.

Method	Dust Table	Scan Bottle	Fold Cloth	Average
π_0	6/20 (30%)	5/20 (25%)	7/32 (22%)	25.6%
HAMLET	5/20 (25%)	5/20 (25%)	6/32 (19%)	22.9%
MemER	8/20 (40%)	7/20 (35%)	12/32 (38%)	37.5%
Uniform sampling	10/20 (50%)	10/20 (50%)	15/32 (47%)	49.0%
IMPRINT	**16/20 (80%)**	**15/20 (75%)**	**23/32 (72%)**	**75.6%**
IMPRINT (OOD)	9/20 (45%)	14/20 (70%)	10/32 (31%)	48.8%

token.

To test the limits of this trend, we additionally train variants with $B = 1$ and $B = 1024$ on RMBench, keeping all other settings identical. At the low end, a single memory token already reaches 67.0%, matching $B = 4$, and attains the highest Press Button success of any capacity (29%). This suggests that a highly compressed memory can still retain coarse counting information, but it underperforms on tasks that require several pieces of information, such as Block Ranking (64%) and Cover Blocks (74%). At the high end, increasing capacity to $B = 1024$ does not yield further gains: average success decreases to 70.0%, below $B = 256$ and comparable to $B = 64$ (70.6%). Relative to $B = 256$, the largest drops occur on Block Ranking (83% to 68%), Swap Blocks (100% to 93%), and Press Button (21% to 15%), while Battery improves (41% to 46%). Notably, $B = 1024$ attains the lowest Stage-1 losses in Table 13 but not the highest task success, reinforcing that reconstruction accuracy alone does not determine control performance. Overall, memory benefits from sufficient capacity without harming tasks that do not rely on memory, but it does not need to grow with the observation history, and excess capacity may leave the representation less constrained without adding task-relevant information. We therefore use $B = 256$ by default.

Stage-1 Supervision-Loss Evaluation. We evaluate checkpoints at 160k updates for $B \in \{1, 4, 16, 64, 256, 1024\}$ on all 450 RMBench validation dataset from policy rollouts in Fig. 7. We deterministically select four uniformly spaced causal prefixes per rollout and 16 historical timesteps per prefix, yielding 1,800 prefixes and 28,800 historical-step queries per checkpoint.

We evaluate the same Stage-1 objective:

$$\mathcal{L}_{\text{mem}} = \mathcal{L}_{\text{vision}} + 0.5\mathcal{L}_{\text{action}} + 0.5\mathcal{L}_{\text{state}} + 0.5\mathcal{L}_{\text{progress}}. \tag{C.1}$$

Visual reconstruction uses 192 DINOv3 patch features per queried timestep, with parameter-free LayerNorm applied independently to predictions and targets. Reconstruction errors are averaged over feature dimensions and queried timesteps. Progress is predicted at each prefix endpoint using the normalized timestep $\rho_t = t/(T-1)$ for a rollout of length T. We average losses over prefixes within each task and equally across the nine tasks, reporting standard errors across task means.

Losses generally decrease with capacity but are non-monotonic (Table 13). Increasing capacity from 4 to 16 tokens raises reconstruction errors despite improving downstream memory-task success (Fig. 7). This is consistent with using reconstruction to learn useful memory representations for policy adaptation, rather than treating reconstruction accuracy as a direct measure of control performance.

Table 13: Stage-1 losses across memory capacities. Total loss is mean ± SEM across tasks; component losses are unweighted. The second header row indicates scaling.

B	Total $\times 10^{-2}$	Vision $\times 10^{-2}$	Action $\times 10^{-4}$	State $\times 10^{-4}$	Progress $\times 10^{-4}$
1	3.649 ± 0.295	3.599	2.37	2.43	5.13
4	3.308 ± 0.273	3.278	1.31	1.35	3.31
16	3.406 ± 0.277	3.368	1.50	1.55	4.41
64	3.389 ± 0.282	3.359	1.50	1.52	2.94
256	3.338 ± 0.280	3.311	1.27	1.31	2.81
1024	3.256 ± 0.270	3.232	1.08	1.10	2.63

C.2.2 Recurrent and Full-attention Encoders

The per-task comparison between the two encoders is included in Tables 10 and 11. On RMBench, the two encoders reach nearly identical average success (72.8% vs. 72.7%) and differ by at most 10 points on any task, with the largest differences on Press Button (31% recurrent vs. 21% full) and Battery (35% vs. 41%). On RoboCasa, the recurrent encoder improves average success from 62.2% to 66.7% and is higher on 8 of 12 tasks, with the largest gains on Place Equal Ice Cubes (20% vs. 6%), Organize Condiments (74% vs. 62%), Scrub Cutting Board (76% vs. 66%), and Wash and Return (96% vs. 86%).

The two encoders also differ in inference cost. The full-attention encoder revisits the entire history at every query, so its cost grows with episode length. The recurrent encoder instead updates a fixed-size state once per chunk, so its cost stays bounded. As a result, the recurrent encoder reduces latency from 130.8 ms to 122.8 ms per action chunk (Fig. 6; detailed inference time analysis in App. C.5) by averaging over the entire trajectory. It also grows more slowly than full attention variant when the horizon length increases. Overall, the recurrent encoder matches the full-attention encoder on RMBench, improves it on RoboCasa, and runs faster, which is why we use it as the default IMPRINT encoder.

C.3 Memory Learning and VLA Backbones

C.3.1 Training Recipe and Stage 1 Objectives

Table 15 shows that Stage 1 training alone is sufficient for some tasks. With frozen memory, Swap T rises from 45% to 99% and Rearrange Blocks from 90% to 99%. Other tasks require joint adaptation. Cover Blocks rises only from 15% with end-to-end training to 35% with frozen memory, but reaches 100% after joint adaptation. This is consistent with the analysis in Fig. 9, where frozen memory already encodes the color order but the policy cannot yet use it. On Battery, Block Ranking, and Press Button, frozen memory is even below end-to-end training, and only joint adaptation improves success.

The memory objective ablation shows how reconstruction supervision from different streams affects task performance. Visual reconstruction is essential for Cover Blocks, whose memory is the initial color order: without it, success drops from 100% to 52%, whereas visual reconstruction alone keeps 100%. Block Ranking follows the same pattern (61% without visual reconstruction vs. 85% with it alone). Conversely, action and state reconstruction benefit Rearrange Blocks (100% vs. 78% with visual reconstruction only) and Press Button (23% vs. 15%). Removing progress estimation mainly hurts multi-stage tasks, lowering Rearrange Blocks to 91%, Swap Blocks to 90%, and Block Ranking to 73%.

C.4 Analysis of the Learned Memory

This section describes the analyses in Fig. 9. All of them use Cover Blocks, where the robot must recall the initial left-to-right order of the red, green, and blue blocks, a six-way classification over their

Table 14: Per-task success as the capacity of the full-attention memory encoder is varied (Fig. 7). All other architecture, training, and evaluation settings are fixed. $B = 1$ and $B = 1024$ are evaluated on RMBench only (–). The best result in each row is shown in bold; rows where all settings tie are left unmarked.

Task	Type	B=1	B=4	B=16	B=64	B=256	B=1024
RMBench							
Observe / pick	Memory	7	8	8	8	9	**10**
Rearrange blocks	Memory	**100**	**100**	**100**	**100**	**100**	99
Put back block	Memory	100	100	100	100	100	100
Swap blocks	Memory	97	97	99	99	**100**	93
Swap T	Memory	99	99	99	99	**100**	99
Battery	Memory	33	40	44	39	41	**46**
Block ranking	Memory	64	73	79	81	**83**	68
Cover blocks	Memory	74	68	83	83	**100**	**100**
Press button	Memory	**29**	18	15	26	21	15
RMBench average	Memory	67.0	67.0	69.7	70.6	**72.7**	70.0
RoboCasa							
Retrieve fruit	Memory	–	80	74	80	**84**	–
Retrieve oil	Memory	–	66	**78**	70	76	–
Wash and return	Memory	–	**92**	84	82	86	–
Wash and return to same location	Memory	–	46	**62**	60	**62**	–
Rinse cutting board	Memory	–	0	0	0	0	–
Sweeten hot chocolate	Memory	–	40	44	54	**58**	–
RoboCasa memory average	Memory	–	54.0	57.0	57.7	**61.0**	–
Pan transfer	Non-memory	–	**78**	68	70	76	–
Frying-pan adjustment	Non-memory	–	58	58	58	**72**	–
Organize condiments	Non-memory	–	48	52	60	**62**	–
Place equal ice cubes	Non-memory	–	4	**10**	6	6	–
Scrub cutting board	Non-memory	–	60	60	56	**66**	–
Counter to sink	Non-memory	–	**98**	94	96	**98**	–
RoboCasa non-memory average	Non-memory	–	57.7	57.0	57.7	**63.3**	–

permutations. We compare three memory encoders: end-to-end training from scratch (E2E), frozen Stage 1 memory (Frozen), and jointly adapted IMPRINT (Ours). For each encoder, we reconstruct the causal history at 64 uniformly spaced timesteps in each of the 50 Cover Blocks demonstrations and pass the visual, action, proprioceptive, and language inputs through the fixed memory encoder. We concatenate the mean- and max-pooled features of its $B = 256$ output tokens to obtain a 2,048-dimensional representation. The six color orders appear in 11, 7, 10, 10, 3, and 9 demonstrations for B–G–R, B–R–G, G–B–R, G–R–B, R–B–G, and R–G–B, respectively. Because these demonstrations were also used to train the memory, the analyses describe the learned representation rather than generalization to new trajectories.

Linear probe. We measure whether the learned latent memory representation automatically include color order information by training a six-way linear softmax classifier on the features, giving 3,200 samples per encoder. We use five-fold cross-validation stratified by class at the episode level, so all timesteps of an evaluation episode are excluded from training. Features are standardized with training-fold statistics, and the classifier is optimized for 300 full-batch gradient steps with a learning rate of 0.2 and ℓ_2 regularization of 10^{-3}. Timestep accuracy scores each sampled prefix, and episode accuracy

Table 15: Per-task success for the training-recipe and Stage 1 objective ablations on RMBench (Fig. 8). The best result in each row is shown in bold separately within each ablation; rows where all variants tie are left unmarked.

	Training recipe			Stage 1 objective			
Task	E2E	IMPRINT frozen	IMPRINT	IMPRINT	Without progress	Visual only	Action + state
Observe / pick	**12**	7	9	**9**	6	6	5
Rearrange blocks	90	99	**100**	**100**	91	78	**100**
Put back block	100	100	100	100	100	100	100
Swap blocks	90	99	**100**	**100**	90	89	89
Swap T	45	99	**100**	**100**	97	99	98
Battery	27	20	**41**	**41**	35	30	36
Block ranking	73	68	**83**	83	73	**85**	61
Cover blocks	15	35	**100**	**100**	**100**	**100**	52
Press button	5	2	**21**	21	**26**	15	23
Average	50.8	58.8	**72.7**	**72.7**	68.7	66.9	62.7

averages the logits over an episode before predicting its class. We also report balanced accuracy to account for the class imbalance. The task-success column reports closed-loop rollouts on Cover Blocks and is not used to train the probe. As shown in Table 17, Stage 1 training raises timestep accuracy from 19.94% to 83.31%, and joint adaptation raises it further to 91.53%.

t-SNE visualization. We visualize only the second half of each episode, closer to the uncovering phase in which the color order guides behavior, which gives 32 points per episode and 1,600 points per encoder. The features are standardized and reduced to 50 dimensions with PCA, and then embedded with t-SNE using Euclidean distance, perplexity 30, PCA initialization, an automatic learning rate, 1,500 iterations, and seed 42. Class labels are used only for coloring and for computing separation afterward. Point color indicates the color-order class, and saturation and opacity increase with episode progress from 50% to 100%.

The mean silhouette coefficient in the two-dimensional embedding is −0.08 for E2E, 0.23 for Frozen, and 0.36 for IMPRINT. In the second half of the episode, the blocks are largely covered, so the color order can no longer be read from the current observation and must come from memory. The E2E memory shows no grouping by color order: its negative silhouette indicates that episodes with different orders are mixed together, consistent with its near-chance probe accuracy (19.94% vs. 16.7% chance). After Stage 1 training alone, the Frozen memory already groups episodes by color order, even though Stage 1 receives neither explicit color-order labels nor gradients from a downstream policy. This shows that reconstructing the multi-stream history is enough for the memory to retain this hidden variable from the early frames. Joint adaptation separates the groups further, indicating that the action objective refines the memory toward the information the policy needs. Together, these results show that the learned memory organizes past experience by the task-relevant variable rather than by the current scene. Because t-SNE is fitted separately for each encoder, only the structure within each panel is comparable, not coordinates across panels. The linear probe provides the quantitative measure of how

Table 16: Per-task success with and without IMPRINT across three VLA backbones on RMBench (Fig. 10). For each backbone, the better result of the two variants is shown in bold; ties are left unmarked.

	$\pi_{0.5}$		π_0		MolmoAct2	
Task	No memory	IMPRINT	No memory	IMPRINT	No memory	IMPRINT
Observe / pick	8	**9**	**8**	6	4	4
Rearrange blocks	63	**100**	13	**99**	52	**100**
Put back block	20	**100**	49	**100**	0	**100**
Swap blocks	16	**100**	4	**100**	11	**65**
Swap T	8	**100**	0	**98**	15	**100**
Battery	30	**41**	27	**30**	**7**	5
Block ranking	66	**83**	54	**56**	62	**74**
Cover blocks	15	**100**	0	**78**	18	**83**
Press button	3	**21**	**2**	0	0	**6**
Average	25.4	**72.7**	17.4	**63.0**	18.8	**59.7**

Table 17: Cover Blocks color-order probes and closed-loop task success (%). Probes are evaluated on held-out episode folds drawn from the memory model's training demonstrations.

Memory training	Timestep accuracy	Balanced accuracy	Episode accuracy	Task success
E2E	19.94	17.29	42.00	15.00
Frozen	83.31	83.68	90.00	35.00
IMPRINT	**91.53**	**85.95**	**98.00**	**100.00**

much color-order information the memory retains.

Attention visualization. At inference step 750, we record the cross-attention weights of the full-attention memory encoder. In each of its two blocks and eight heads, these weights connect the 256 latent queries to history tokens. Averaging over blocks, heads, and queries yields one attention score per history token. We then assign the valid tokens to five streams: head-camera, left-wrist, and right-wrist observations, actions, and proprioceptive states. Visual memory tokens are mapped back to their temporal chunk and camera view, while action and state tokens keep their original timestep. Scores are ranked separately within each stream and displayed from timestep 0 to 750 with a purple-to-red colormap, where warmer colors indicate higher attention within that stream. In the displayed episode, Frozen and IMPRINT place substantial attention on the initial head-camera frames that reveal the color order, whereas E2E emphasizes proprioceptive history. This visualization shows which inputs the memory attends to in one episode and does not establish their causal contribution. The recurrent encoder has no cross-attention over history tokens and would require a separate attribution method.

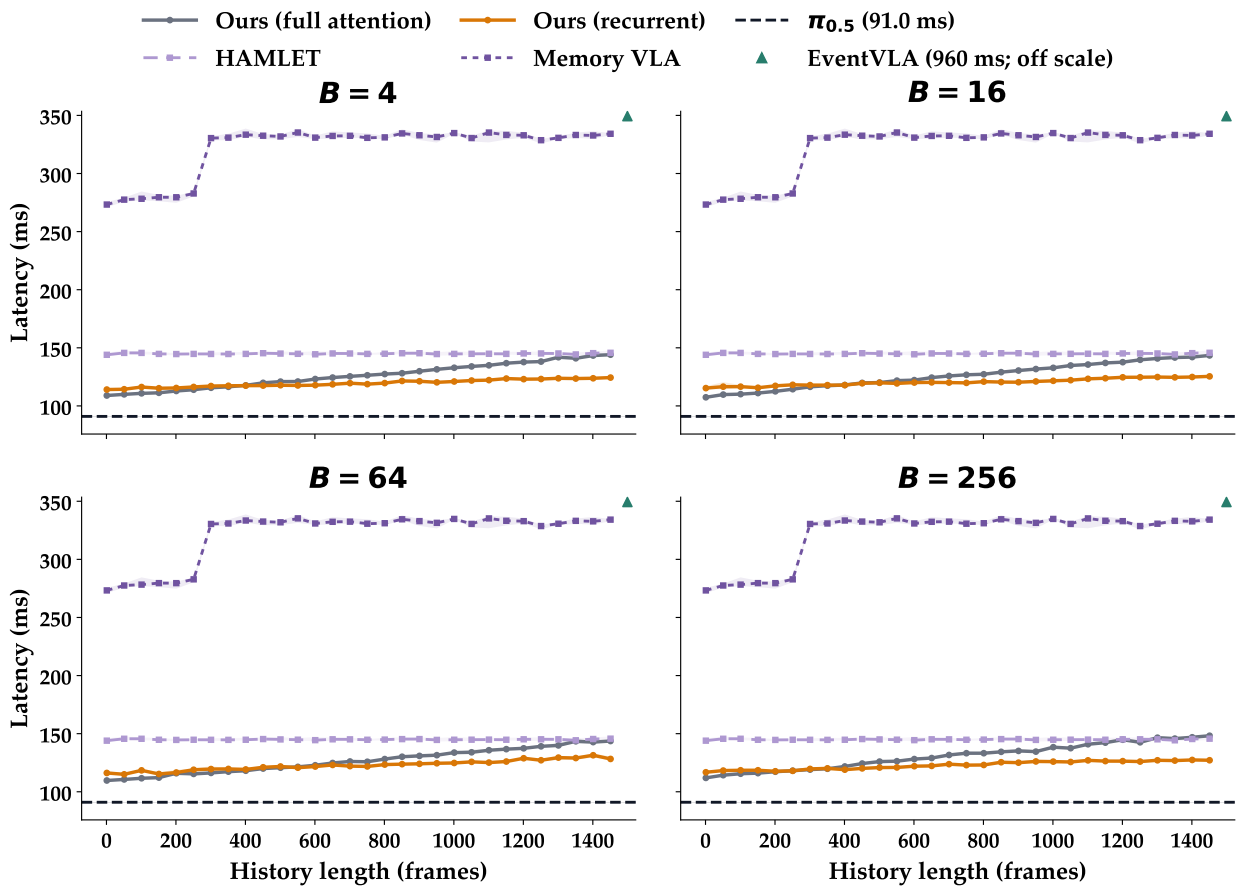


Figure 15: **Inference Time Analysis For Different Bottleneck Size and Different Methods**

Table 18: Policy-call latency in milliseconds at the shortest and longest history. All variants of our method predict 50 actions.

	Full attention		Recurrent	
B	1 frame	1,451 frames	1 frame	1,451 frames
4	109.0	144.1	114.1	124.4
16	107.4	143.4	115.3	125.4
64	109.7	143.7	116.2	128.3
256	111.9	148.2	116.8	127.1

C.5 Inference-Time Analysis

Protocol. We measure the synchronized wall-clock latency of one policy call on an NVIDIA L40S GPU with batch size one. Unless stated otherwise, each call predicts a 50-step action chunk. We vary the available history from 1 to 1,451 environment frames in increments of 50 frames. Curves report median latency, and shaded regions denote the interquartile range. Environment stepping, video recording, and the subsequent history-buffer update are excluded. For our method, recent-image features are computed once per history state and reused across timing repetitions, matching the steady-state cached evaluation path. The full result of runtime analysis is in Fig. 15.

Comparison with existing methods. The non-memory $\pi_{0.5}$ policy requires 91.0 ms per 50-action prediction, providing a lower bound corresponding to the underlying policy without memory. At $B = 256$, our recurrent memory requires 116.8 ms at the shortest history and 127.1 ms at 1,451 frames. Thus, incorporating long-term memory introduces a 36.1 ms overhead over $\pi_{0.5}$ at the longest history, while retaining a sub-130 ms policy latency.

HAMLET requires approximately 144–146 ms throughout the episode. Its moment-token extraction, rolling-memory update, memory Transformer, and 50-action generation are all included in this measurement. HAMLET uses a fixed $K = 8$ rolling window with a stride of 50 frames. Its cache contents are updated at every policy call, but the number of processed moment tokens remains constant; consequently, its latency does not increase with episode length. At 1,451 frames, our recurrent $B = 256$ model is approximately 18.6 ms, or 12.8%, faster than HAMLET.

Memory VLA is substantially more expensive, increasing from 273.3 ms to 334.2 ms. It predicts a 50-step action chunk, so we report its measured latency without rescaling. The abrupt increase between 251 and 301 frames occurs when its 256-entry memory becomes full. Every subsequent update activates token-merging consolidation, increasing latency from approximately 283 ms to 330 ms. Memory VLA is approximately 2.6× slower than our recurrent model at the longest history. EventVLA has a reported latency of 960 ms and is shown off-scale for reference.

Effect of memory capacity. Table 18 reports latency as the number of memory tokens B varies. These measurements slice the query bank of the same $B = 256$ variant; they isolate inference cost and are not independently trained capacity variants.

Increasing B from 4 to 256 adds only 4.1 ms for full attention and 2.7 ms for recurrent memory at the longest history. The relationship is not strictly monotonic because these differences are small relative to runtime variation and fixed-shape policy computation. Memory capacity therefore has a much smaller effect on latency than the mechanism used to process temporal history.

Scaling with history horizon. Full attention grows by 34–36 ms between 1 and 1,451 frames, corresponding to a fitted slope of approximately 24–25 ms per 1,000 frames. Depending on B, this represents

a 31–34% increase. The recurrent variants grow by only 10–12 ms, with slopes of approximately 6.6–9.7 ms per 1,000 frames and relative increases of 8.8–10.4%.

The difference is explained by the history representations. Full attention constructs an input whose active history grows from approximately 194 to 4,008 tokens. At $B = 256$, history preparation consequently increases from 1.1 to 33.7 ms. In contrast, recurrent memory incrementally compresses completed history into a fixed-size state and reads only that state together with a short recent window. Its history-preparation cost increases from 1.0 to only 8.5 ms. Action generation itself remains nearly constant at approximately 79–82 ms for both architectures.

Full attention is slightly faster for very short histories because recurrent memory pays a fixed update and read overhead. The recurrent implementation becomes faster after approximately 250–550 frames, depending on B. At the longest history it reduces latency by 15.4–21.1 ms, or 10.7–14.2%, relative to full attention. These results show that recurrent compression provides its main computational advantage as temporal context grows, while adding only modest overhead over the non-memory policy.

D Additional Ablation Results

These comparisons isolate choices in the visual representation and in the historical input streams supplied to the memory encoder. Table 6 defines the corresponding configurations.

D.1 Ablations on Visual Encoder For Input Stream & Supervision Objective

Frame-level visual features enter memory learning at two points: the recent visual stream, which covers frames not yet encoded by the video tokenizer, and the Stage 1 visual reconstruction target. We compare two feature choices that change both points together. The *DINOv3* variant uses the same DINOv3 features for the recent stream and the reconstruction target, so the encoder receives the same representation that the decoder learns to predict. Each image is encoded by a frozen DINOv3 ViT-B/16 at 256×256 resolution, and its patch grid is average-pooled to 64 tokens of width 768 per camera view. The *SigLIP* variant instead takes the recent stream from the SigLIP vision encoder of the policy($\pi_{0.5}$) and uses the image-token hidden states of its PaliGemma language model as the reconstruction target, each pooled to 64 tokens of width 2048 per view. Unlike the DINOv3 variant, the input and target are therefore different representations. All other components, including the memory bottleneck, the action and proprioceptive streams, the training objectives, and the evaluation protocol, are held fixed. We do not evaluate mixed configurations, such as DINOv3 inputs with SigLIP targets.

Results. Table 19 shows that IMPRINT performs well with both feature choices, with a small overall advantage for DINOv3. On RoboCasa, DINOv3 improves memory-task success from 57.7% to 61.0%, while both variants reach 63.3% on non-memory tasks, giving a 1.7-point gain in overall success (62.2% vs. 60.5%). On RMBench, DINOv3 improves average success by 2.1 points (72.7% vs. 70.6%). The largest gains occur on Sweeten Hot Chocolate (+10) and Wash and Return to Same Location (+8) on RoboCasa, and on Block Ranking (+11) and Battery (+9) on RMBench. These tasks require tracking changes in object arrangement or state, which is consistent with dense patch features being better suited to capturing local visual changes. SigLIP is better on a few tasks, most notably Press Button (−7) and Observe and Pick Up (−3) on RMBench and Rinse Cutting Board (−4) on RoboCasa, so the advantage is not uniform across tasks. Because the two variants change the recent stream and the reconstruction target together, this comparison measures their combined effect and does not isolate the contribution of either one. Since DINOv3 performs better on average and does not depend on the policy's visual encoder, we use it as the default for all experiments in Sec. 5.

Table 19: Effect of the visual features used for both the recent visual stream and the Stage 1 reconstruction target. *DINOv3* uses pooled DINOv3 patch features for both; *SigLIP* uses pooled SigLIP features as input and PaliGemma image-token hidden states as the target. Entries are per-task success rates (%), and Δ is DINOv3 minus SigLIP. The better result in each row is shown in bold, and ties are bolded for both variants.

(a) RoboCasa

Task	DINOv3	SigLIP	Δ
memory			
Retrieve fruit	**84**	78	+6
Retrieve oil	**76**	**76**	0
Wash and return	**86**	**86**	0
Wash and return to same location	**62**	54	+8
Rinse cutting board	0	**4**	−4
Sweeten hot chocolate	**58**	48	+10
Average	**61.0**	57.7	+3.3
Non-memory			
Pan transfer	**76**	**76**	0
Frying-pan adjustment	72	**74**	−2
Organize condiments	**62**	60	+2
Place equal ice cubes	6	**8**	−2
Scrub cutting board	66	**68**	−2
Counter to sink	**98**	94	+4
Average	**63.3**	**63.3**	0
Overall average	**62.2**	60.5	+1.7

(b) RMBench

Task	DINOv3	SigLIP	Δ
Observe/pickup	9	**12**	−3
Rearrange blocks	**100**	98	+2
Put back block	**100**	**100**	0
Swap blocks	**100**	93	+7
Swap T	**100**	**100**	0
Battery	**41**	32	+9
Blocks ranking	**83**	72	+11
Cover blocks	**100**	**100**	0
Press button	21	**28**	−7
Overall average	**72.7**	70.6	+2.1

D.2 Ablation on Historical Input Streams

The Stage-1 objective ablation in Sec. 5.4 removes *supervision* for individual streams while still encoding all three streams. Here we instead remove the *inputs* themselves, to measure how much each history stream contributes to memory. We fix the full-attention encoder at $B = 256$ and keep visual history, recent DINOv3 features, and language conditioning in all variants. *Vision only* removes both action and proprioceptive inputs and their reconstruction objectives. *Vision + Action* retains executed actions and action reconstruction but removes proprioception. *Vision + State* retains proprioception and state reconstruction but removes actions. Disabled streams are excluded from the encoder's key–value sequence, and their reconstruction losses are not computed. All other settings match the default configuration.

Results. Table 20 shows that both non-visual streams contribute, and that the full combination performs best (72.7%). Among the ablated variants, Vision + Action is strongest at 68.4%, compared with 60.6% for Vision only and 60.9% for Vision + State. Relative to visual history alone, adding actions improves Swap T (45% to 99%), Swap Blocks (81% to 100%), Press Button (7% to 18%), and Battery (30% to 38%), tasks in which the relevant history consists of interaction events such as which blocks have been moved or how many presses have occurred. These events are difficult to infer from temporally sampled visual observations alone.

Adding proprioception without actions leaves the average nearly unchanged, but this masks large offsetting per-task effects. Vision + State matches Vision + Action on Swap T (99%) and nearly on Swap Blocks (98%), yet sharply degrades Block Ranking (80% to 46%) and Cover Blocks (100% to 70%). One possible explanation is that proprioceptive history alone offers an easy but incomplete cue,

Table 20: Historical input-stream ablation on RMBench with the full-attention encoder at $B = 256$. Each variant removes the listed streams from both the memory input and the Stage-1 reconstruction targets. The best result in each column is shown in bold; columns where all variants tie are left unmarked.

Historical inputs	Obs./pick	Rearrange	Put back	Swap blocks	Swap T	Battery	Ranking	Cover	Press	Avg.
Vision only	**9**	93	100	81	45	30	80	**100**	7	60.6
Vision + Action	6	**100**	100	**100**	99	38	72	83	18	68.4
Vision + State	**9**	94	100	98	99	28	46	70	4	60.9
Vision + Action + State (IMPRINT)	**9**	**100**	100	**100**	**100**	**41**	**83**	**100**	**21**	**72.7**

which the memory may rely on at the expense of visual information; this is consistent with the E2E variant in Fig. 9 attending mainly to state history on Cover Blocks. In contrast, combining state with actions and vision improves over Vision + Action by 4.3 points, with the largest gains on Cover Blocks (83% to 100%) and Block Ranking (72% to 83%). This suggests that the streams are complementary when supervised jointly.

Finally, removing streams from the input is more harmful than removing only their supervision. Encoding only visual history yields 60.6%, whereas encoding all three streams with visual reconstruction only (Table 15) yields 66.9%, with Swap T rising from 45% to 99%. Access to action and proprioceptive history therefore helps even without explicit reconstruction targets for those streams, and supervising them provides a further gain.

E Per-Task Failure Analysis

We analyze representative rollouts to examine errors in task-state tracking and physical execution. Memory-related errors involve incorrect target identities, action orders, repetition counts, or remembered locations and orientations. Manipulation errors involve failed grasps, dropped objects, or inaccurate placements despite selecting the intended target. All comparisons below refer to the illustrated rollouts.

E.1 RMBench Analysis

For each task, we compare IMPRINT with HAMLET and Uniform Sampling, followed by a comparison with the Frozen and End-to-End training variants. Together, these examples illustrate challenges in recalling task-relevant information and executing the corresponding actions precisely.

Battery Try. In the baseline comparison (Fig. 16), IMPRINT completes the task. Both baselines insert one battery but knock the remaining battery off the table during grasping. The training-recipe comparison (Fig. 17) reveals additional failure modes: Frozen completes an insertion but stalls before the required rotation, while End-to-End pushes a battery away and cannot complete the task. In contrast, IMPRINT inserts both batteries and performs the required reorientation. These examples illustrate the need for both correct action sequencing and precise manipulation.

Block Ranking Try. After repeated rearrangement, IMPRINT places the blocks in the target configuration (Fig. 18). The baselines either exceed the time limit or place blocks inaccurately. In the training-recipe comparison (Fig. 19), Frozen presses the button prematurely, while End-to-End fails to reconstruct the target configuration. Only IMPRINT completes the illustrated task, which requires tracking changes in the arrangement and accurately reproducing the resulting configuration.

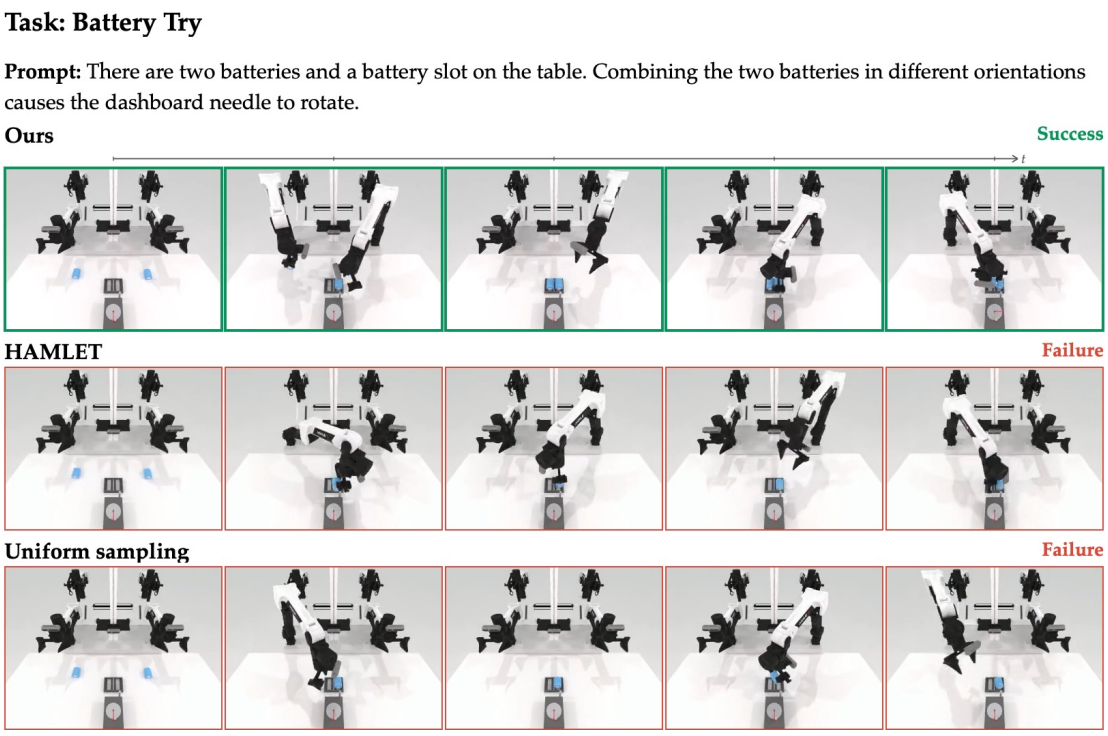


Figure 16: **RMBench: Battery Try.** Representative rollouts of IMPRINT, HAMLET, and Uniform Sampling.

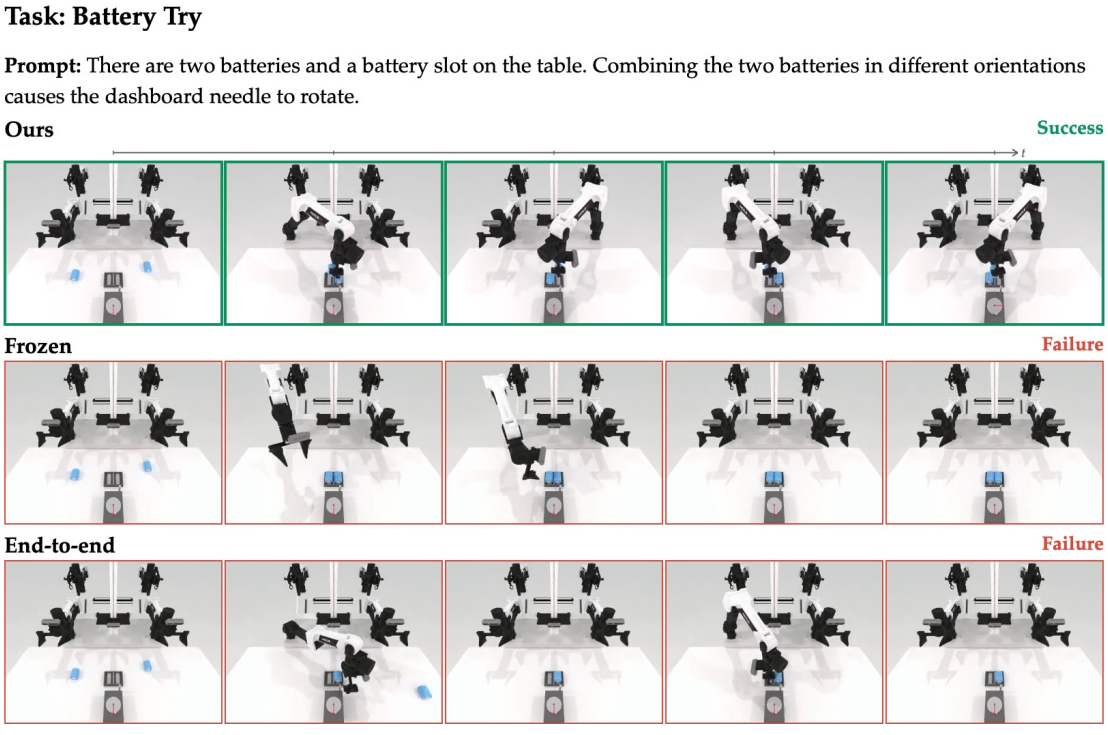


Figure 17: **RMBench: Battery Try.** Representative rollouts of IMPRINT and the Frozen and End-to-End training variants.

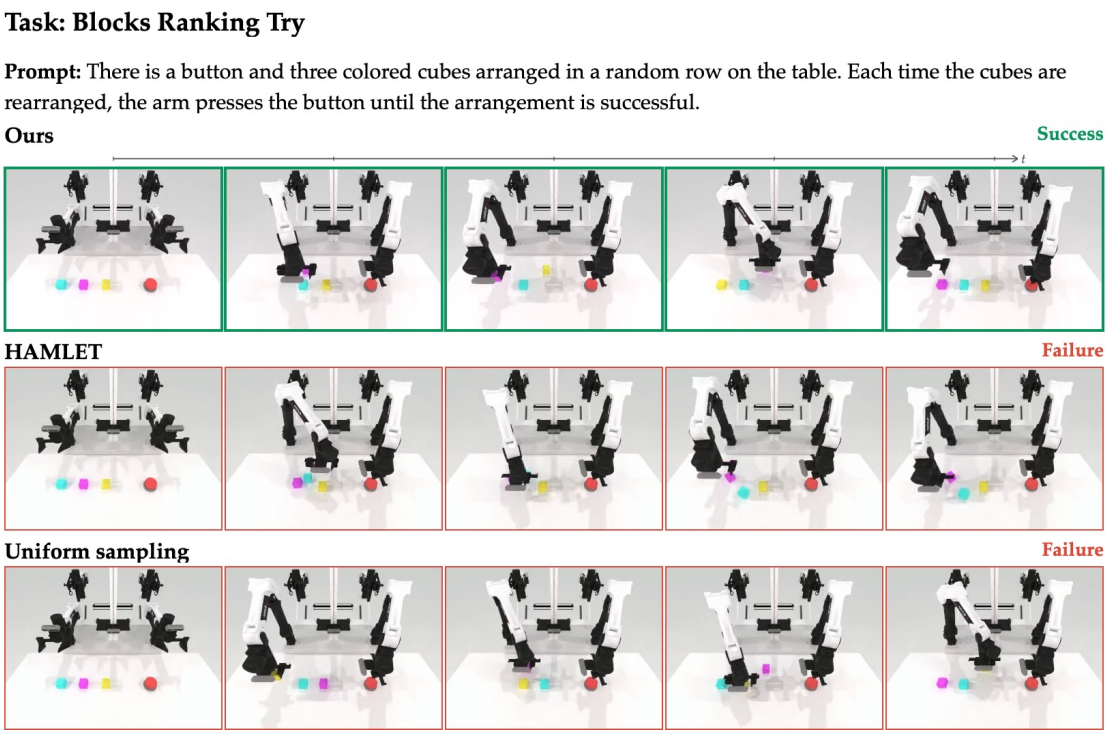


Figure 18: **RMBench: Block Ranking Try.** Representative rollouts of IMPRINT, HAMLET, and Uniform Sampling.

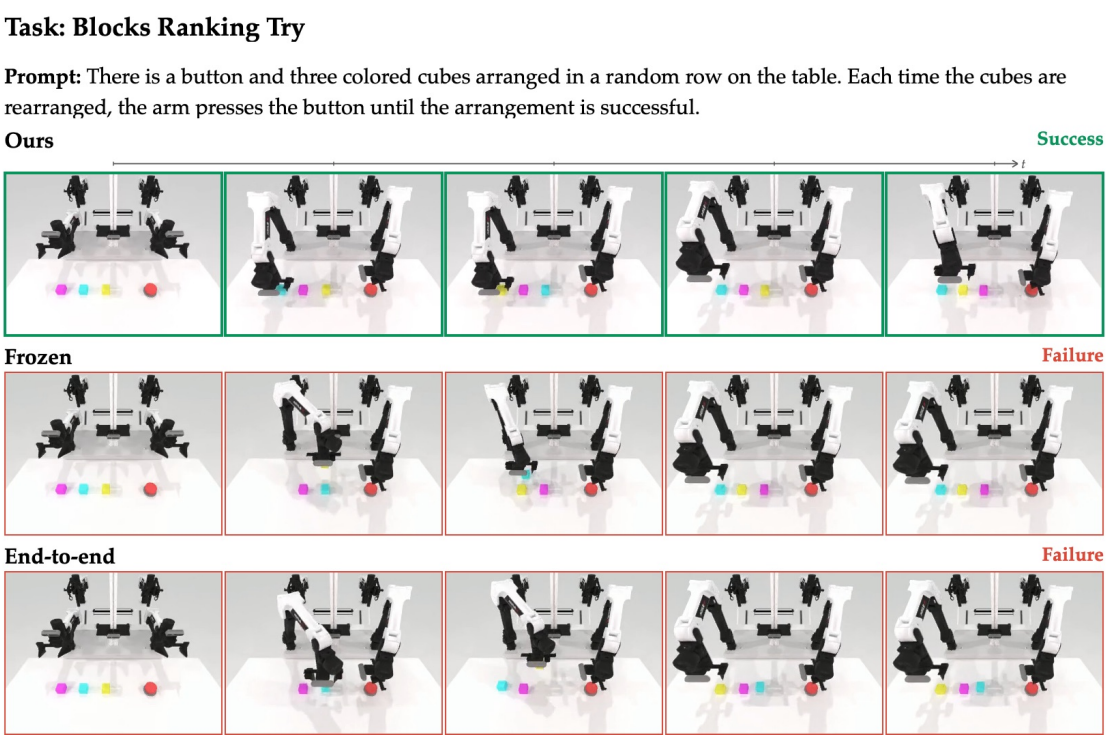


Figure 19: **RMBench: Block Ranking Try.** Representative rollouts of IMPRINT and the Frozen and End-to-End training variants.

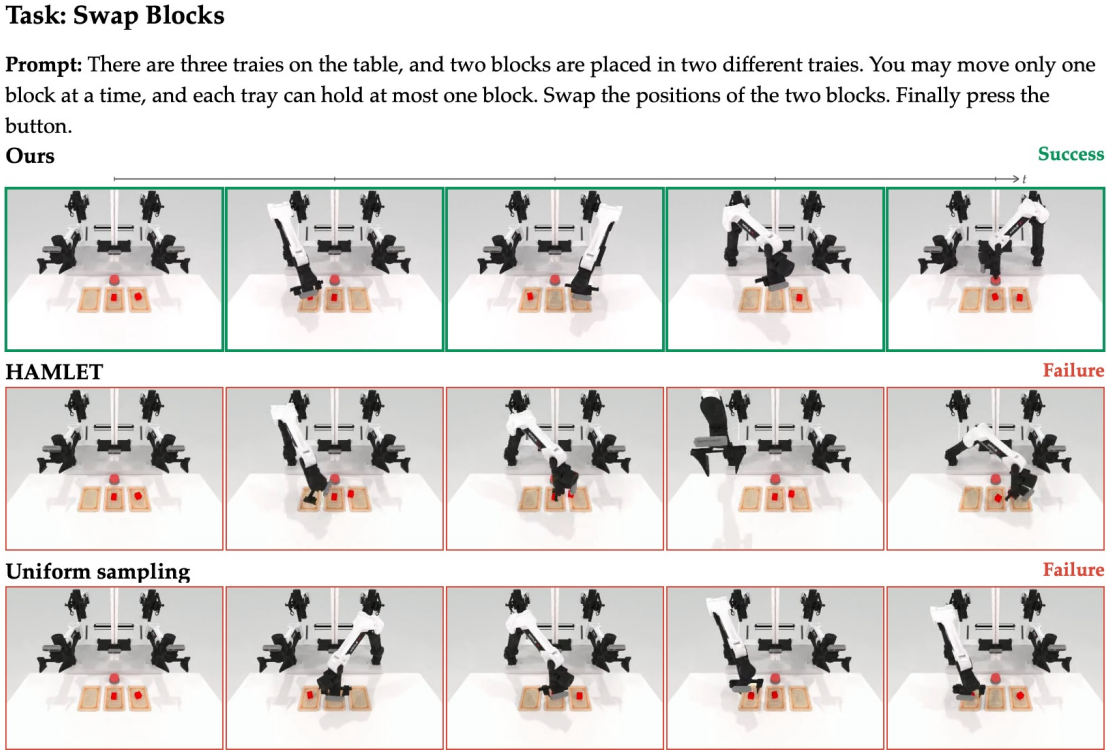


Figure 20: **RMBench: Swap Blocks.** Representative rollouts of IMPRINT, HAMLET, and Uniform Sampling.

Swap Blocks. As shown in Fig. 20, IMPRINT uses the vacant slot on the left as a temporary buffer and swaps the blocks into the correct slots. HAMLET makes an imprecise placement, while Uniform Sampling follows an incorrect action sequence and exceeds the time limit. In the training-recipe comparison (Fig. 21), both Frozen and End-to-End fail to place the blocks correctly in the required slots, whereas IMPRINT completes the swap.

Swap T. In Fig. 22, IMPRINT swaps the T-shaped blocks and places them within the required position and orientation tolerances. Both baselines exchange the blocks but fail to achieve sufficiently precise final placements. In Fig. 23, Frozen swaps the blocks but leaves the blue block at an incorrect position and orientation, while End-to-End fails to complete the swap. IMPRINT completes both the exchange and the precise final placement.

Put Back Block. In the baseline comparison (Fig. 24), both IMPRINT and Uniform Sampling return the block to the correct location, while HAMLET selects the wrong location. In the training-recipe comparison (Fig. 25), all three variants return the block correctly and press the button.

Cover Blocks. In Fig. 26, IMPRINT uncovers the blocks in the required red–green–blue order. HAMLET instead follows a green–blue–red order, while Uniform Sampling repeatedly covers and uncovers the red block. In the training-recipe comparison (Fig. 27), both Frozen and End-to-End incorrectly uncover the blue block first, whereas IMPRINT follows the required order. These rollouts illustrate difficulties in retaining the order and locations of previously covered blocks.

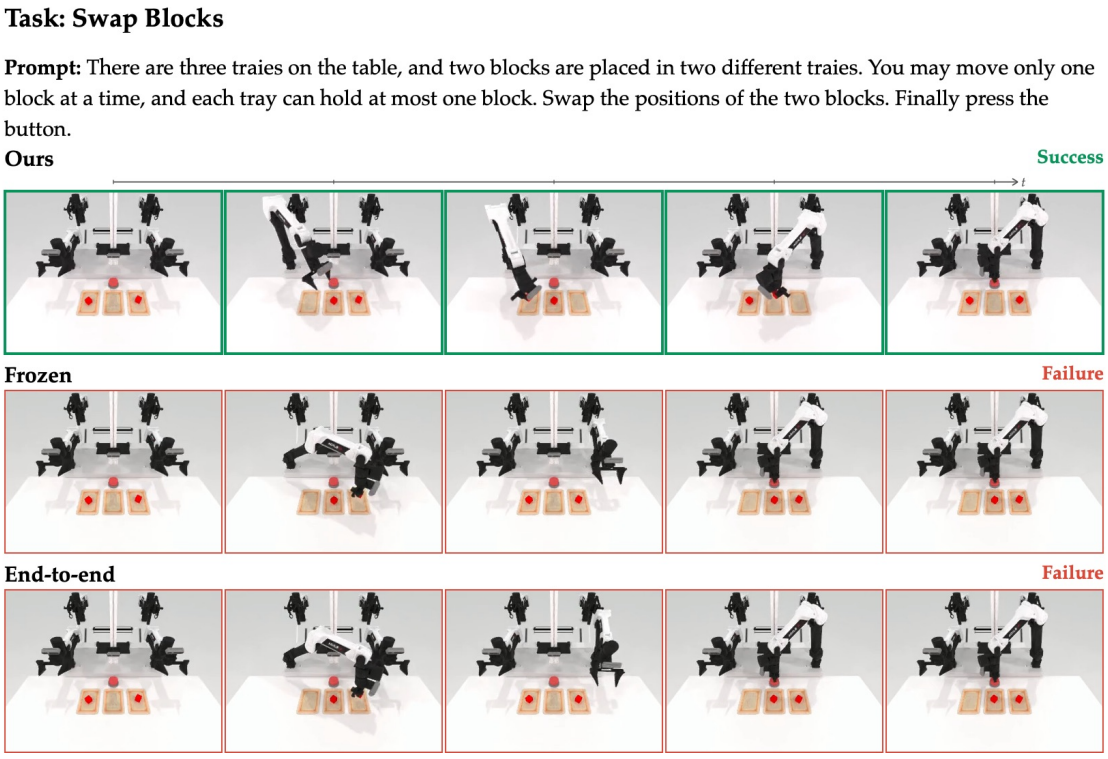


Figure 21: **RMBench: Swap Blocks.** Representative rollouts of IMPRINT and the Frozen and End-to-End training variants.

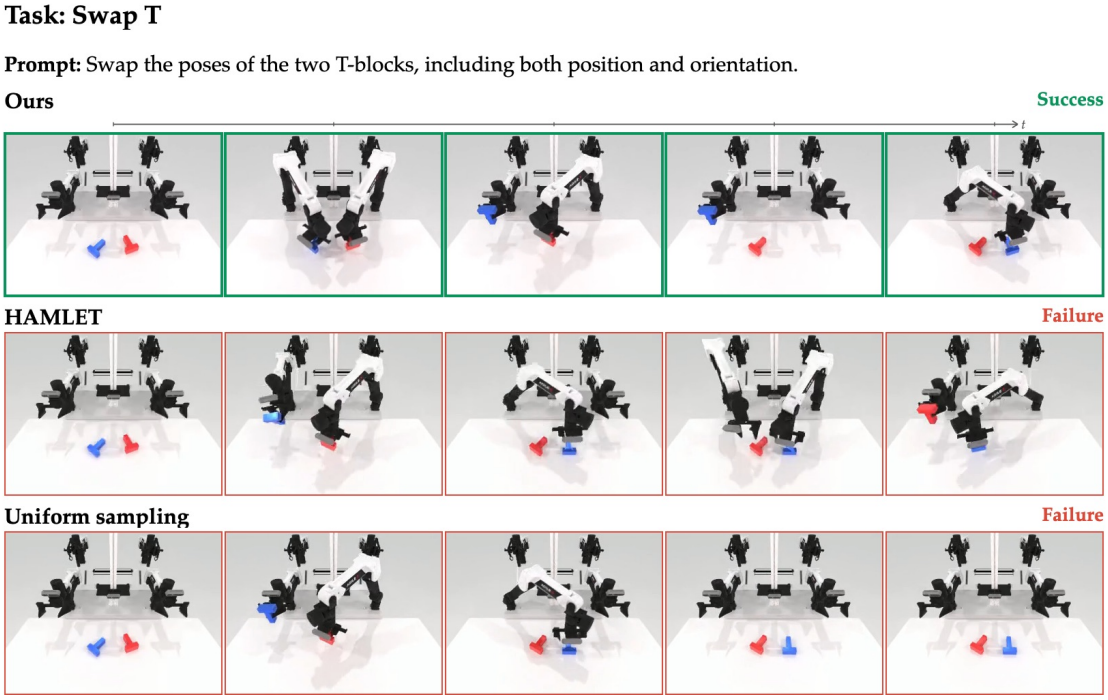


Figure 22: **RMBench: Swap T.** Representative rollouts of IMPRINT, HAMLET, and Uniform Sampling.

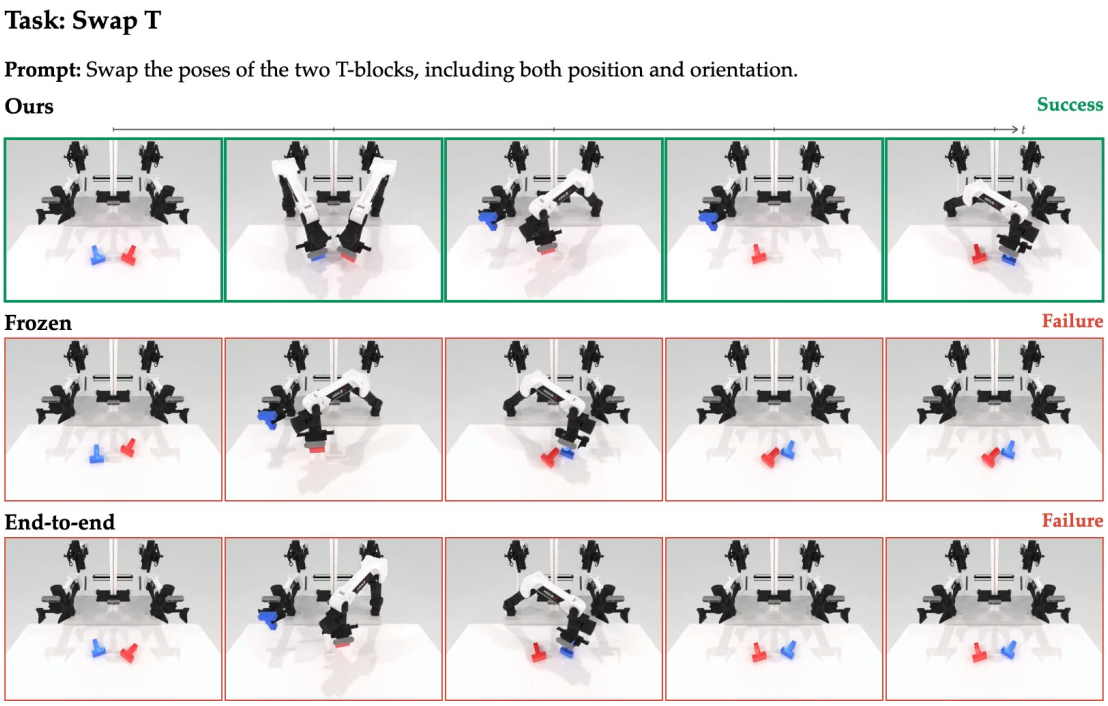


Figure 23: **RMBench: Swap T.** Representative rollouts of IMPRINT and the Frozen and End-to-End training variants.

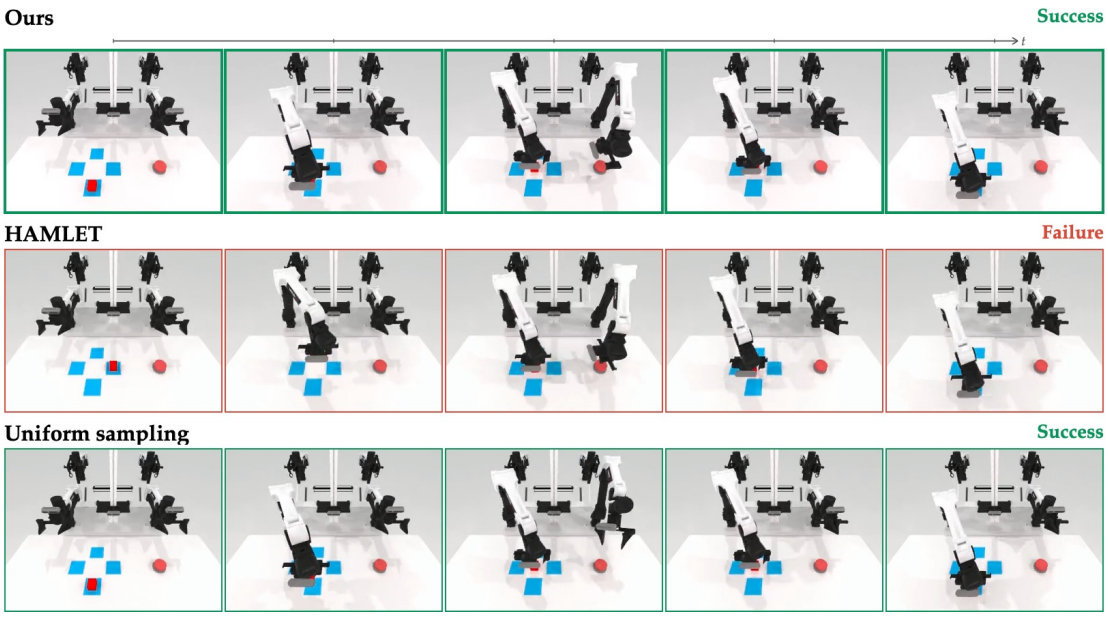


Figure 24: **RMBench: Put Back Block.** Representative rollouts of IMPRINT, HAMLET, and Uniform Sampling.

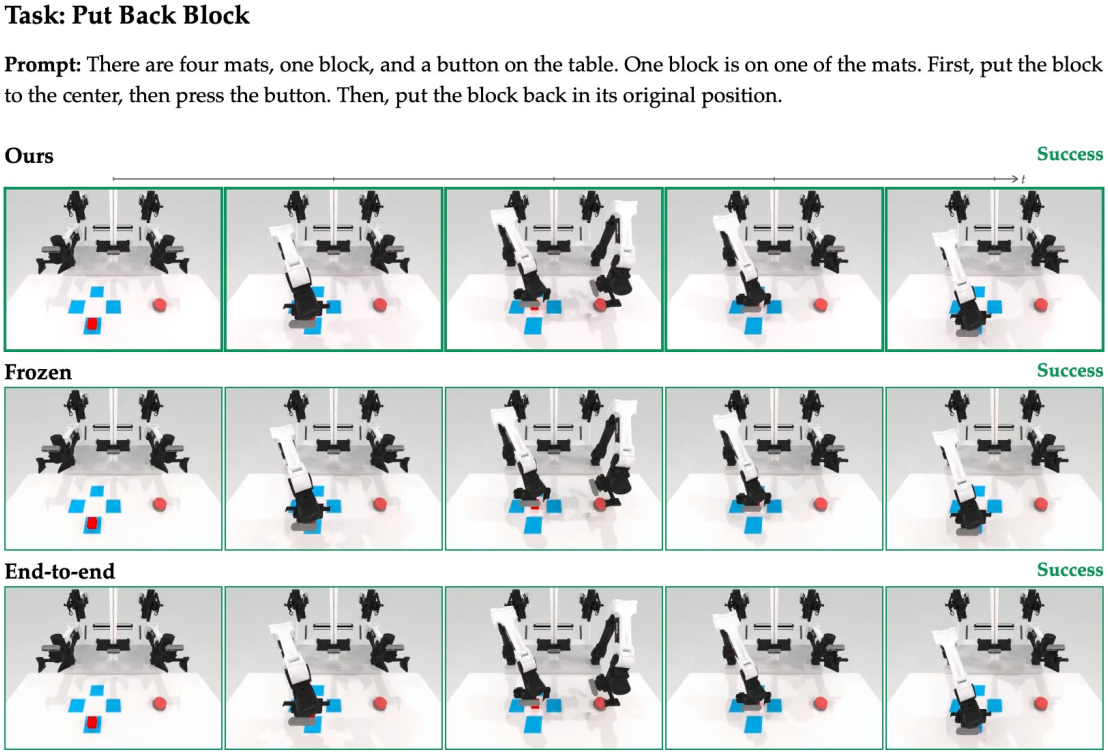


Figure 25: **RMBench: Put Back Block.** Representative rollouts of IMPRINT and the Frozen and End-to-End training variants.

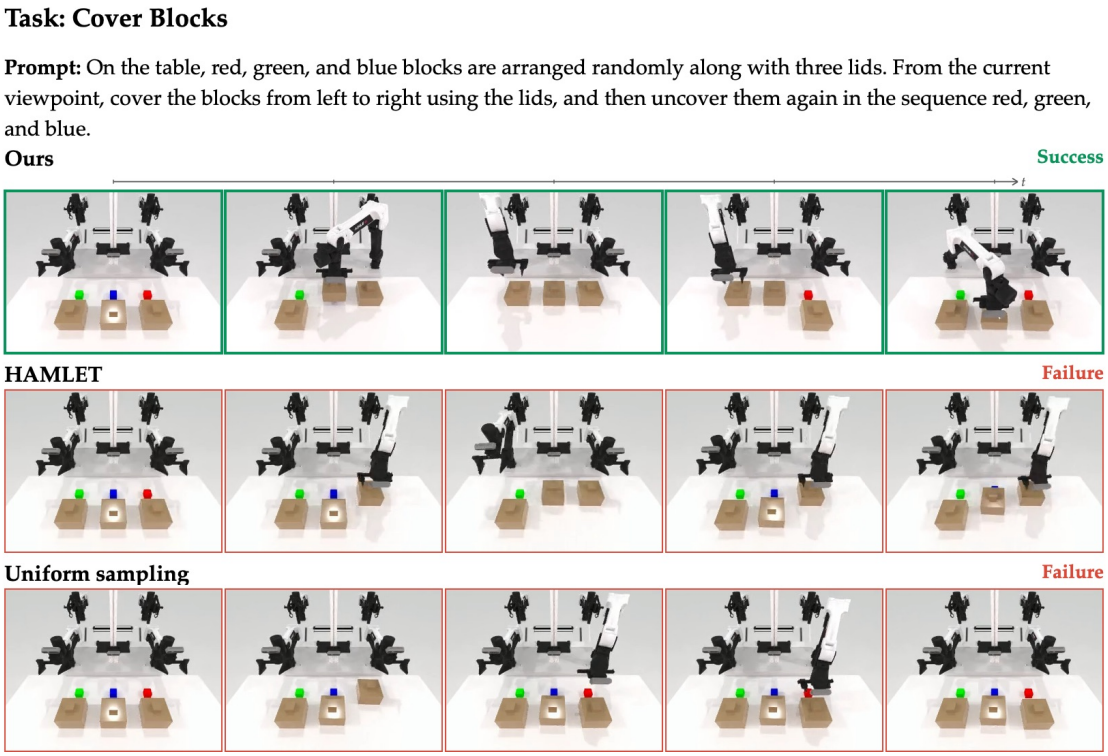


Figure 26: **RMBench: Cover Blocks.** Representative rollouts of IMPRINT, HAMLET, and Uniform Sampling.

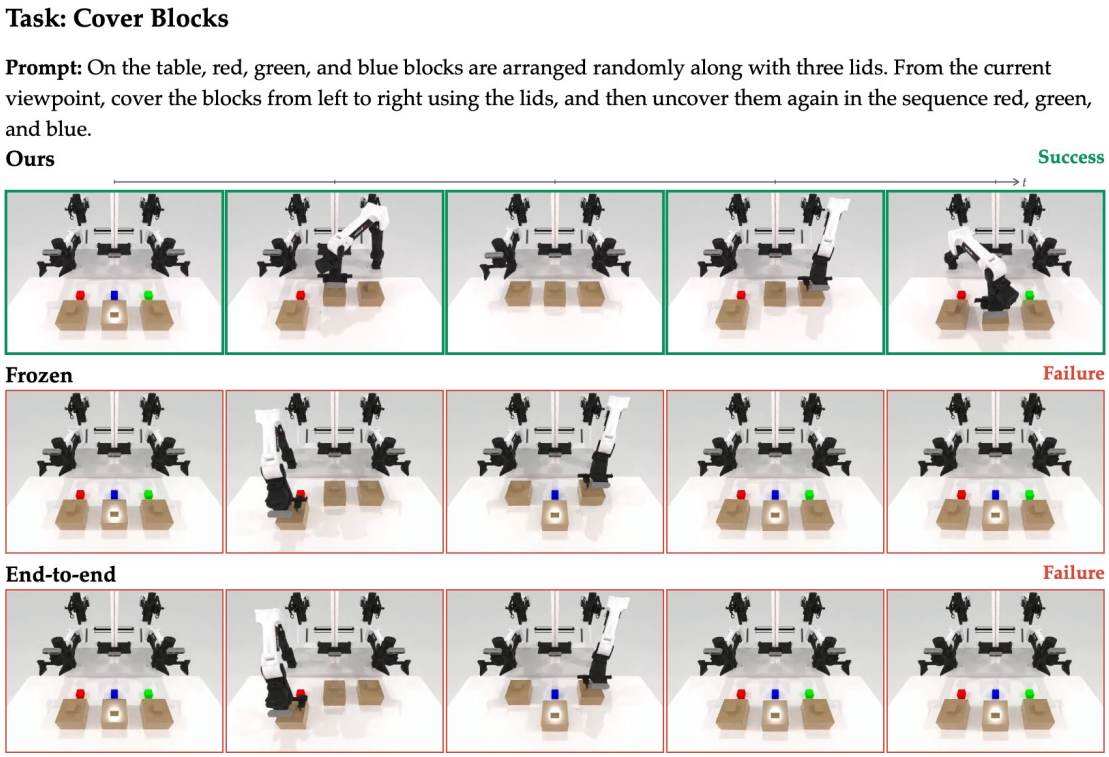


Figure 27: **RMBench: Cover Blocks.** Representative rollouts of IMPRINT and the Frozen and End-to-End training variants.

Press Button. In Fig. 28, IMPRINT completes the required eight presses on the right button and one press on the left. Uniform Sampling presses the right button only once, and HAMLET presses it only twice. In the separate training-recipe example (Fig. 29), IMPRINT completes two presses on the left button and eight on the right, while both ablations fail to press the right button enough times. These examples illustrate errors in tracking and completing the required repetition counts.

Observe and Pick Up. In Fig. 30, IMPRINT grasps the target object shown during the initial observation, while both baselines fail to locate the correct object. Similarly, in the training-recipe comparison (Fig. 31), only IMPRINT successfully picks up the target object; Frozen and End-to-End fail to retrieve it. These examples concern retaining the identity and location of an object observed earlier in the episode.

Rearrange Blocks. In the baseline comparison (Fig. 32), only IMPRINT returns the blocks to the target configuration and releases the gripper. Uniform Sampling baseline follows the correct sequence and places both blocks correctly. It fails because the right gripper remains closed around the second block instead of releasing. In the training-recipe comparison (Fig. 33), Frozen approaches the target arrangement but places a block inaccurately, while End-to-End fails to achieve the required arrangement. IMPRINT completes the rearrangement with accurate final placements.

E.2 RoboCasa Analysis

We compare IMPRINT with $\pi_{0.5}$ and two Uniform Sampling baselines that use VLM-context or action-expert conditioning. The rollouts illustrate errors in remembering locations, tracking counts and task

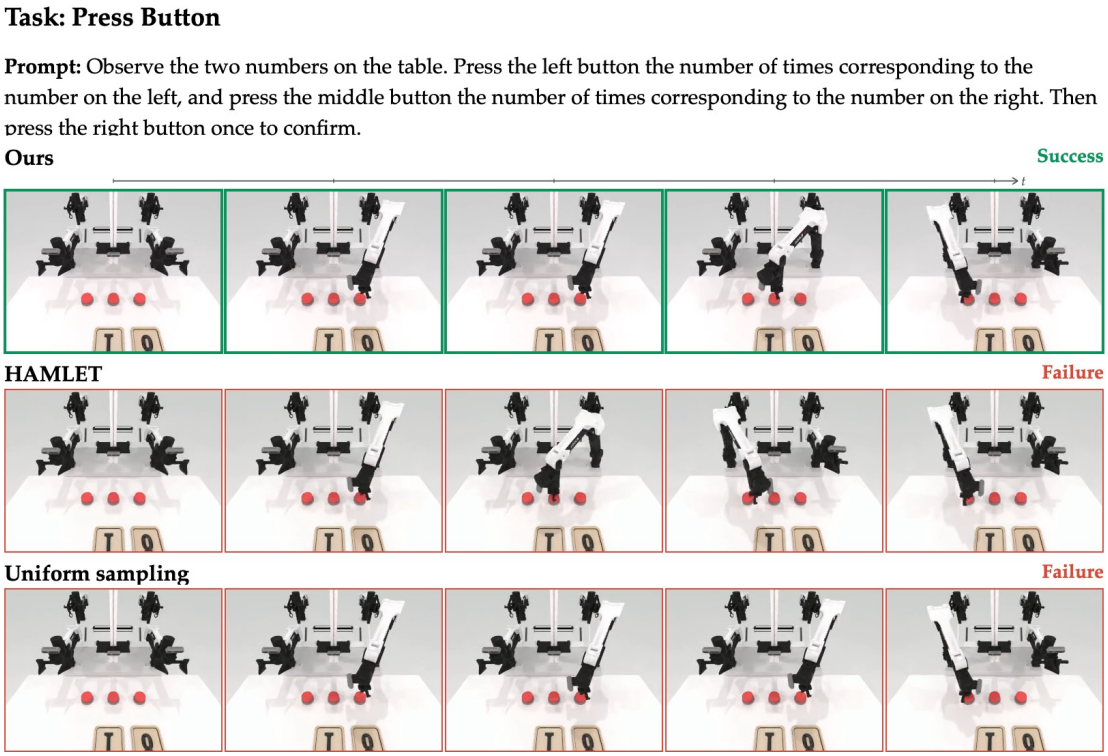


Figure 28: **RMBench: Press Button.** Representative rollouts of IMPRINT, HAMLET, and Uniform Sampling.

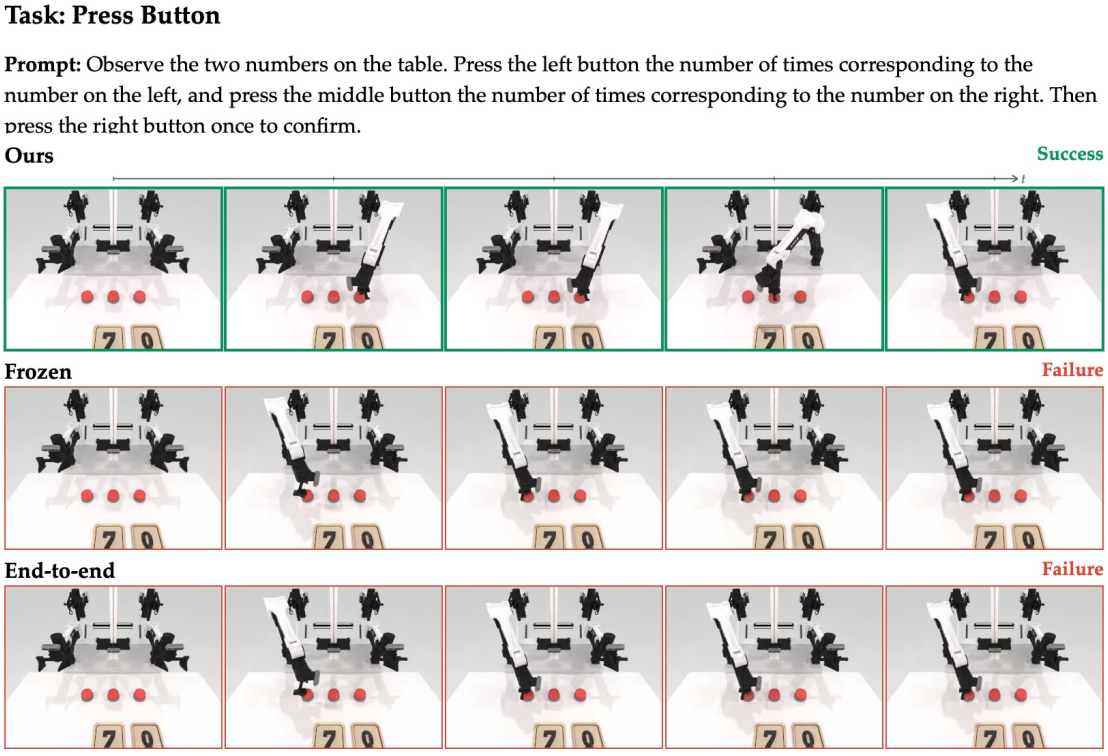


Figure 29: **RMBench: Press Button.** Representative rollouts of IMPRINT and the Frozen and End-to-End training variants.

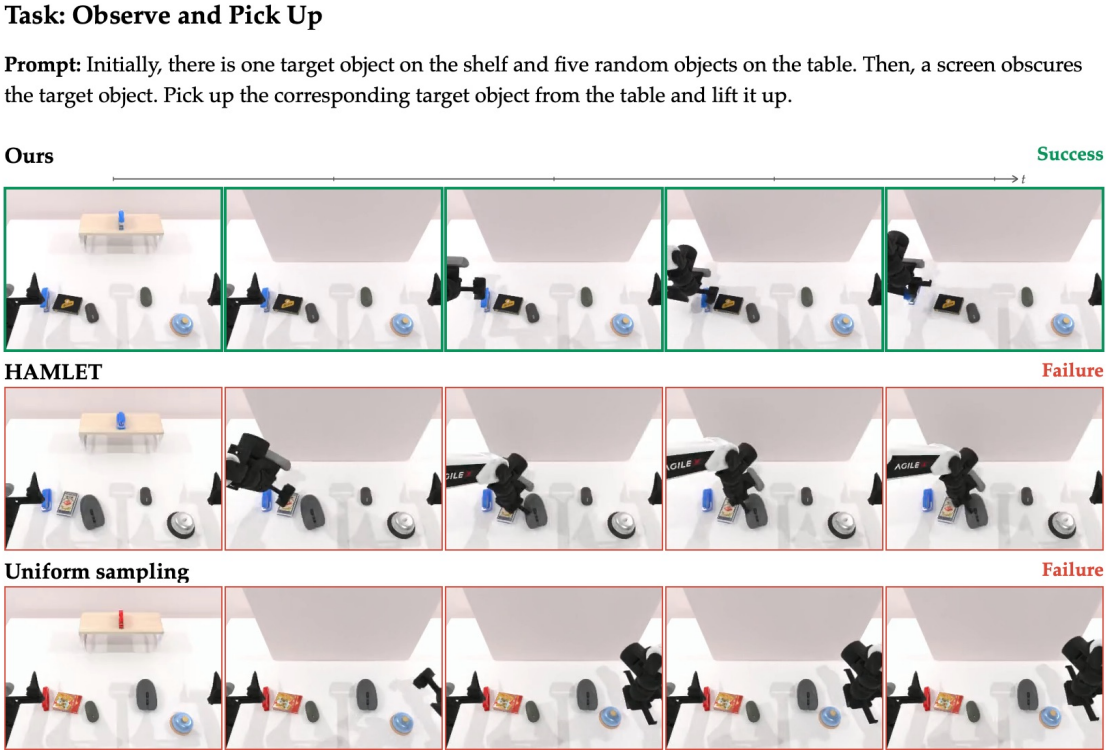


Figure 30: **RMBench: Observe and Pick Up.** Representative rollouts of IMPRINT, HAMLET, and Uniform Sampling.

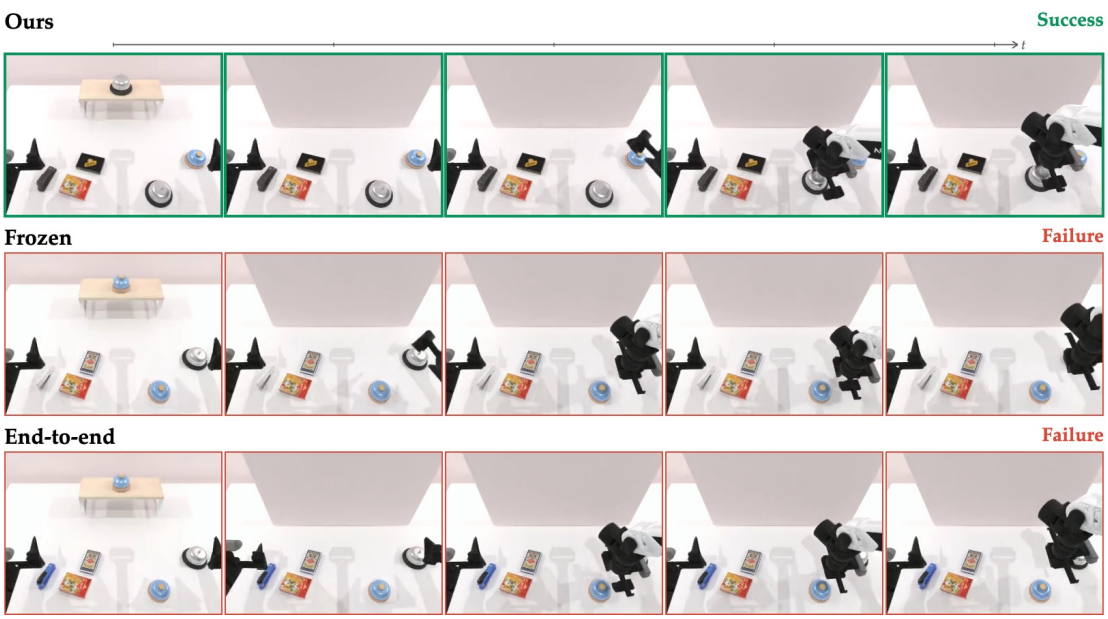


Figure 31: **RMBench: Observe and Pick Up.** Representative rollouts of IMPRINT and the Frozen and End-to-End training variants.

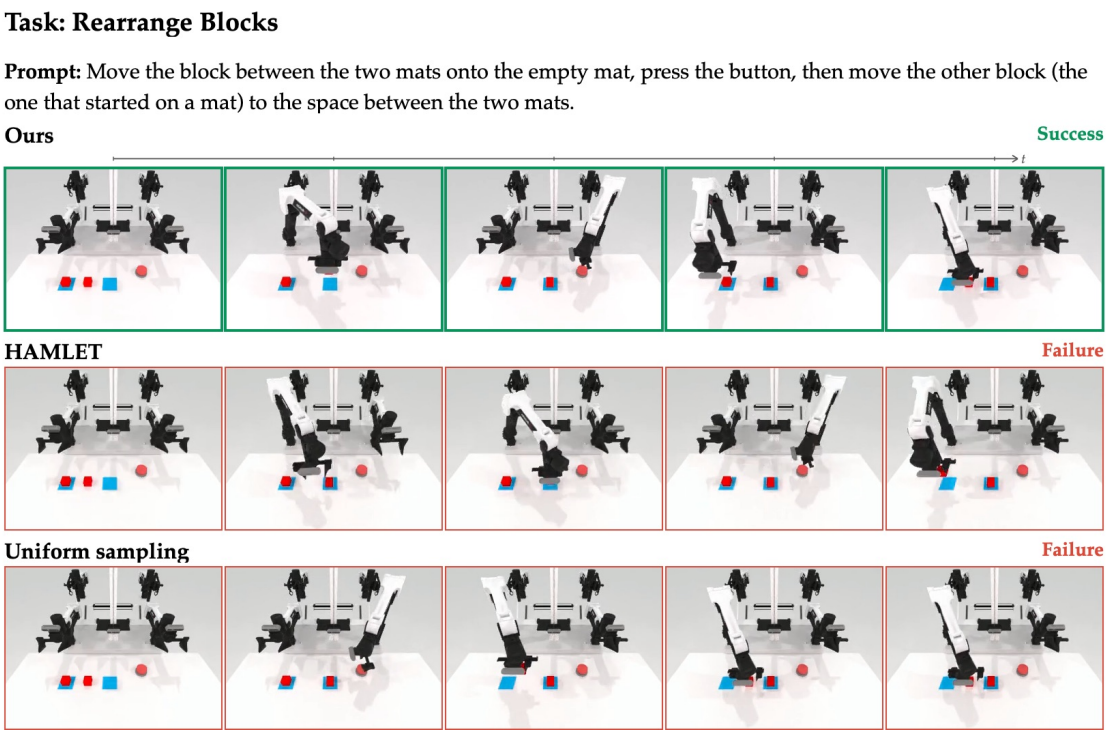


Figure 32: **RMBench: Rearrange Blocks.** Representative rollouts of IMPRINT, HAMLET, and Uniform Sampling.

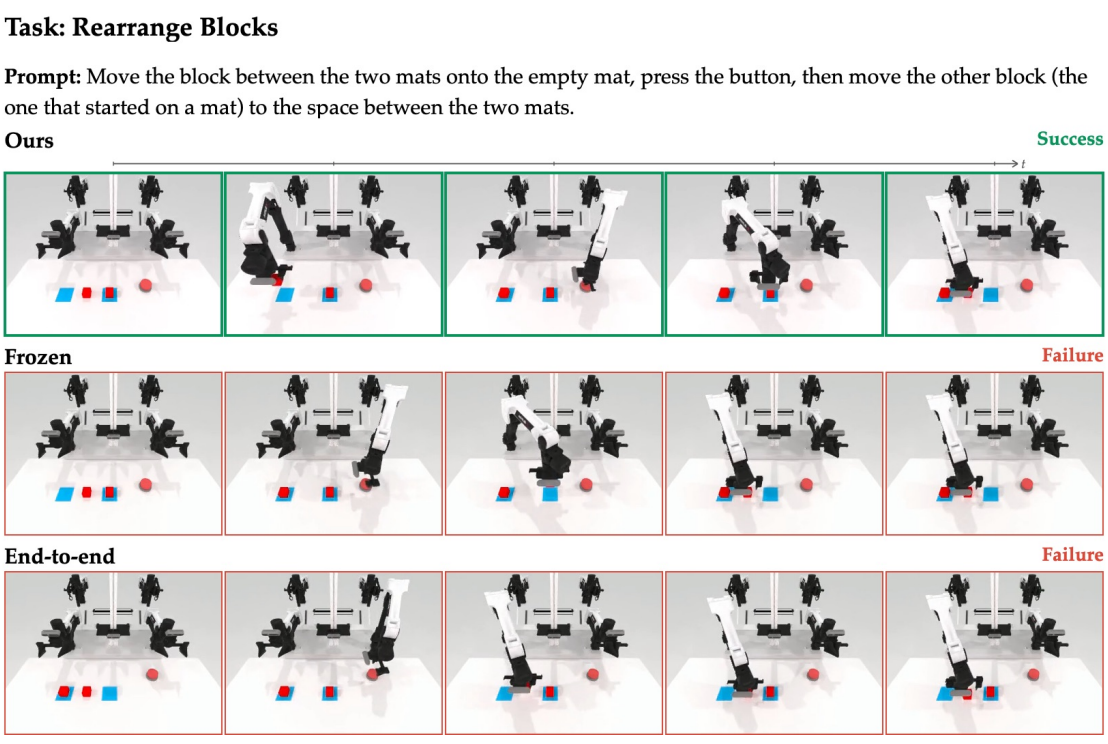


Figure 33: **RMBench: Rearrange Blocks.** Representative rollouts of IMPRINT and the Frozen and End-to-End training variants.

progress, and executing precise interactions. They also include tasks without explicit memory requirements.

Frying Pan Adjustment. As shown in Fig. 34, IMPRINT repositions the frying pan and turns on the stove. All other methods fail to turn on the stove, leaving the task incomplete.

Sweeten Hot Chocolate. In Fig. 35, IMPRINT places exactly one sugar cube in the pot and returns to the reset pose. $\pi_{0.5}$ fails to place the cube inside the pot, while both Uniform Sampling baselines add more than one cube. The illustrated failures therefore include both inaccurate placement and an incorrect number of repetitions.

Pan Transfer. Both IMPRINT and $\pi_{0.5}$ transfer the food to the plate and return the pan to its original position on the stove (Fig. 36). Both Uniform Sampling baselines fail to transfer the food successfully and to return the pan to the correct stove position.

Wash and Return to the Original Plate. In Fig. 37, only IMPRINT returns the banana to its original plate after washing. Both Uniform Sampling baselines fail to pick the banana back up from the sink, while $\pi_{0.5}$ returns it to the wrong plate. This example distinguishes a retrieval failure from an incorrect choice of return location.

Wash and Return to the Same Location. In Fig. 38, IMPRINT returns the banana sufficiently close to its original location to satisfy the task's placement tolerance. The other methods either place it too far from the original location or repeatedly pick it up without completing the return.

Rinse Cutting Board. In Fig. 39, IMPRINT turns on the faucet, keeps it running for more than 10 s, and then turns it off. Uniform Sampling with VLM-context conditioning turns on the faucet but fails to turn it off. Uniform Sampling with action-expert conditioning and $\pi_{0.5}$ fail to turn it on. Thus, the illustrated failures occur at different stages of the required interaction sequence.

Scrub Cutting Board. As shown in Fig. 40, IMPRINT and Uniform Sampling with action-expert conditioning scrub a sufficiently large area of the cutting board. Uniform Sampling with VLM-context conditioning and $\pi_{0.5}$ fail to achieve the required coverage.

Organize Condiments. In Fig. 41, IMPRINT places all three condiment items in the cabinet. Neither Uniform Sampling baseline places an item inside the cabinet. $\pi_{0.5}$ places two items inside but leaves the black item behind, failing to complete the full set of transfers.

Pick and Place to Sink. Both IMPRINT and $\pi_{0.5}$ successfully pick up the spoon and place it in the sink, while both Uniform Sampling baselines fail (Fig. 42). This example shows that the qualitative differences extend to a task without an explicit memory requirement.

Retrieve Fruit. In Fig. 43, only IMPRINT retrieves the fruit and places it in the sink. Uniform Sampling with VLM-context conditioning and $\pi_{0.5}$ approach the wrong side of the workspace. Uniform Sampling with action-expert conditioning approaches the correct side but drops the fruit following an imprecise grasp. This comparison illustrates distinct errors in selecting the retrieval location and executing the grasp.

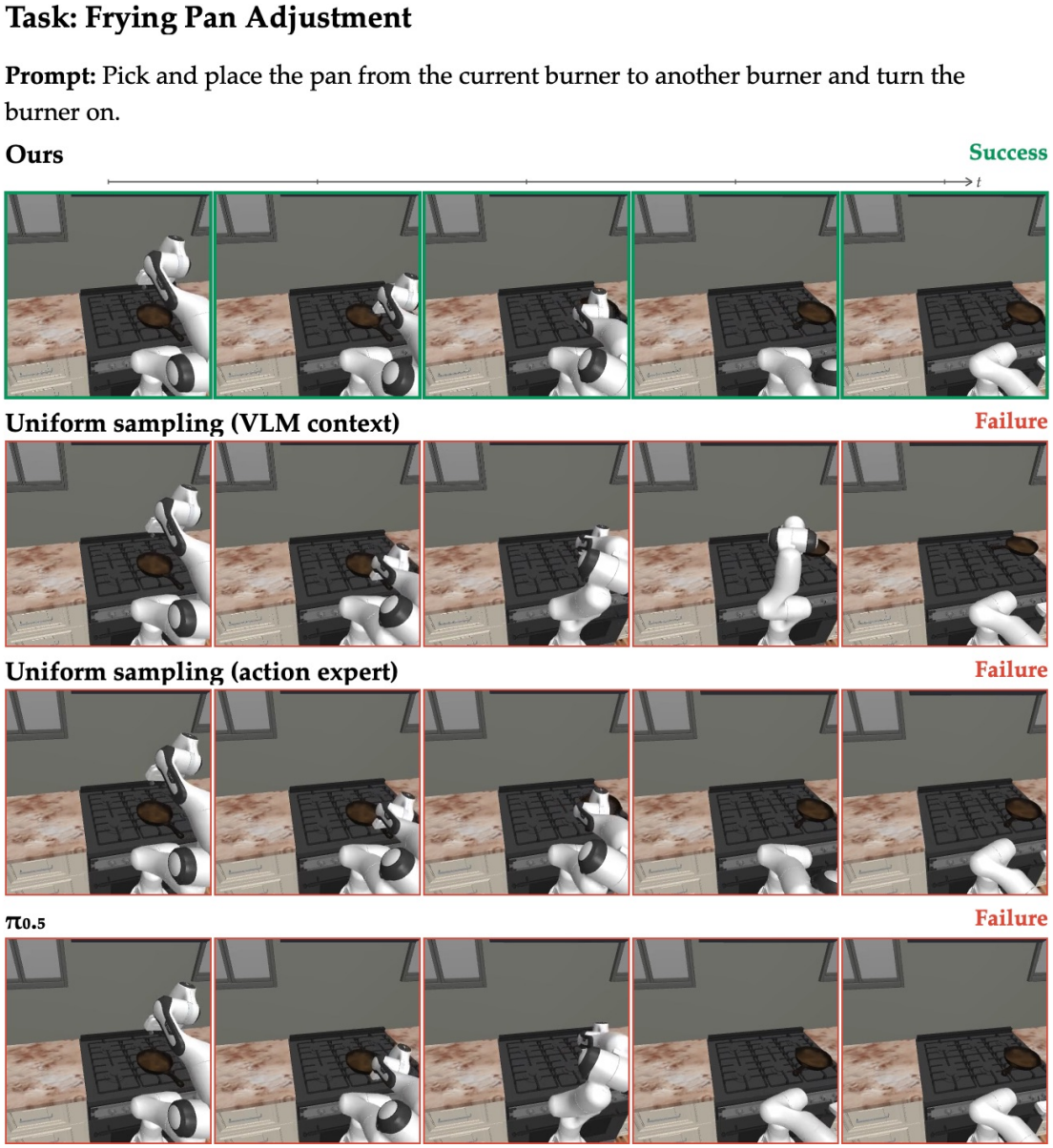


Figure 34: **RoboCasa: Frying Pan Adjustment.** Representative rollouts of IMPRINT, $\pi_{0.5}$, and Uniform Sampling with VLM-context or action-expert conditioning.

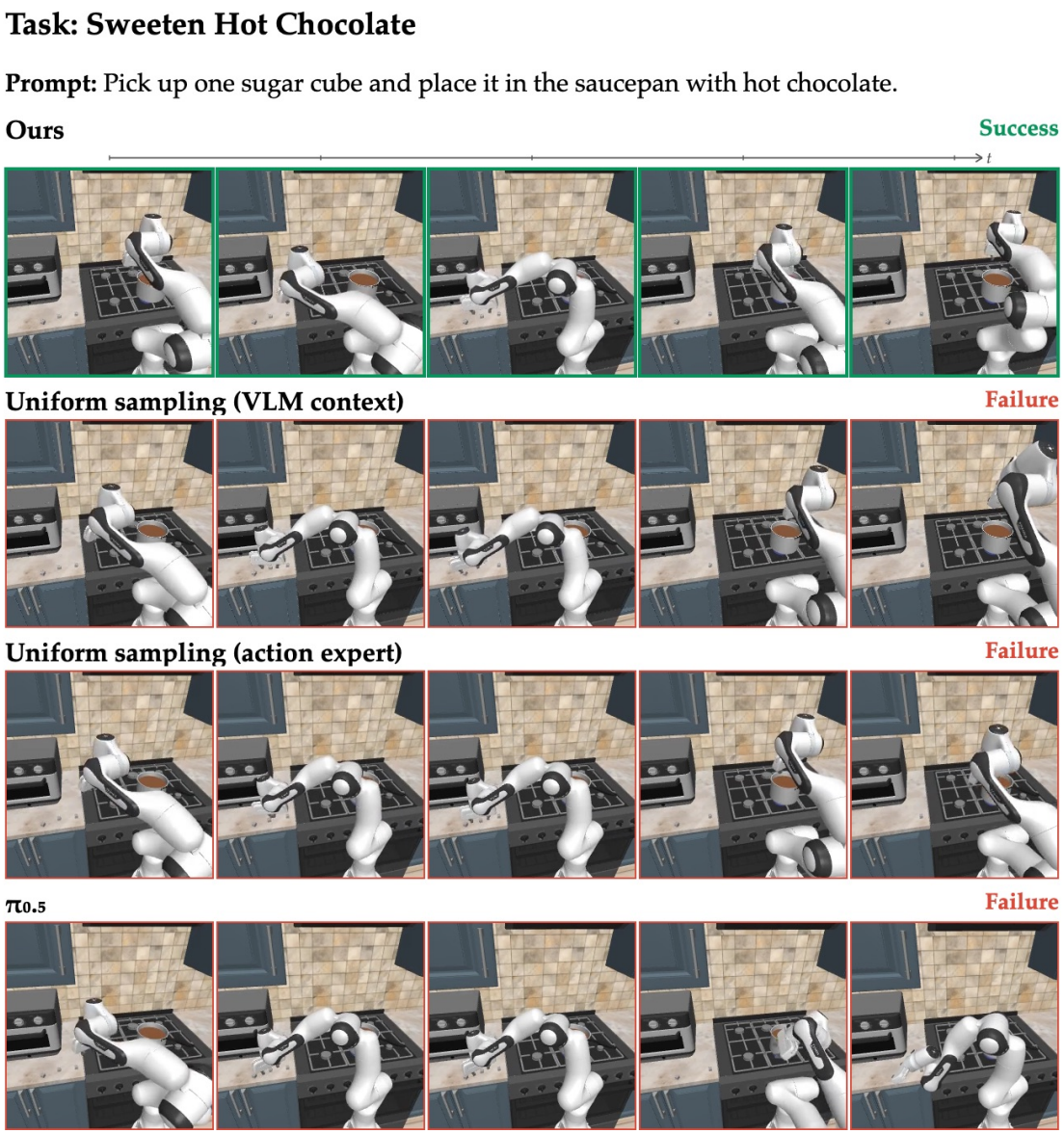


Figure 35: **RoboCasa: Sweeten Hot Chocolate.** Representative rollouts of IMPRINT, $\pi_{0.5}$, and Uniform Sampling with VLM-context or action-expert conditioning.

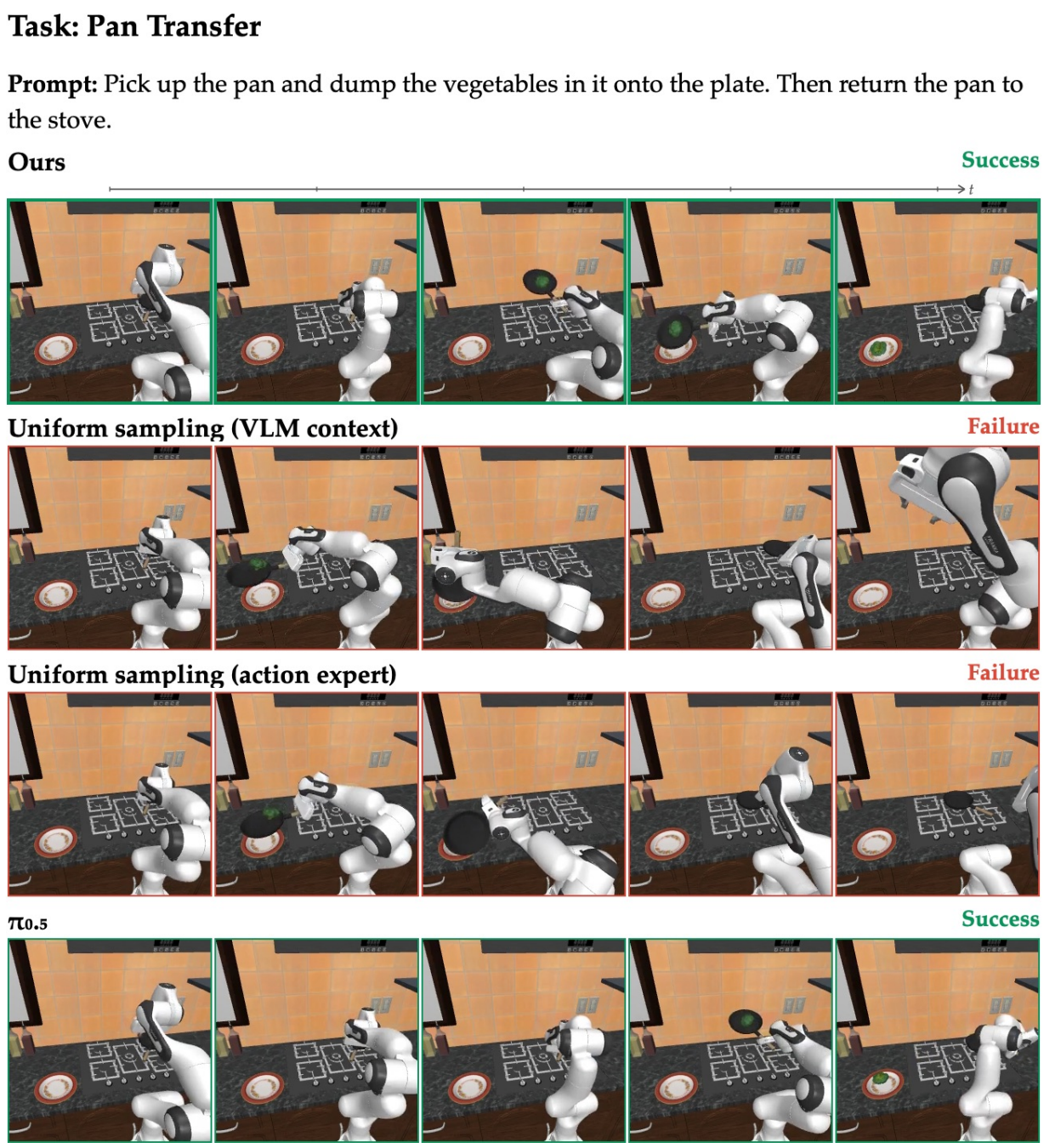


Figure 36: **RoboCasa: Pan Transfer.** Representative rollouts of IMPRINT, $\pi_{0.5}$, and Uniform Sampling with VLM-context or action-expert conditioning.

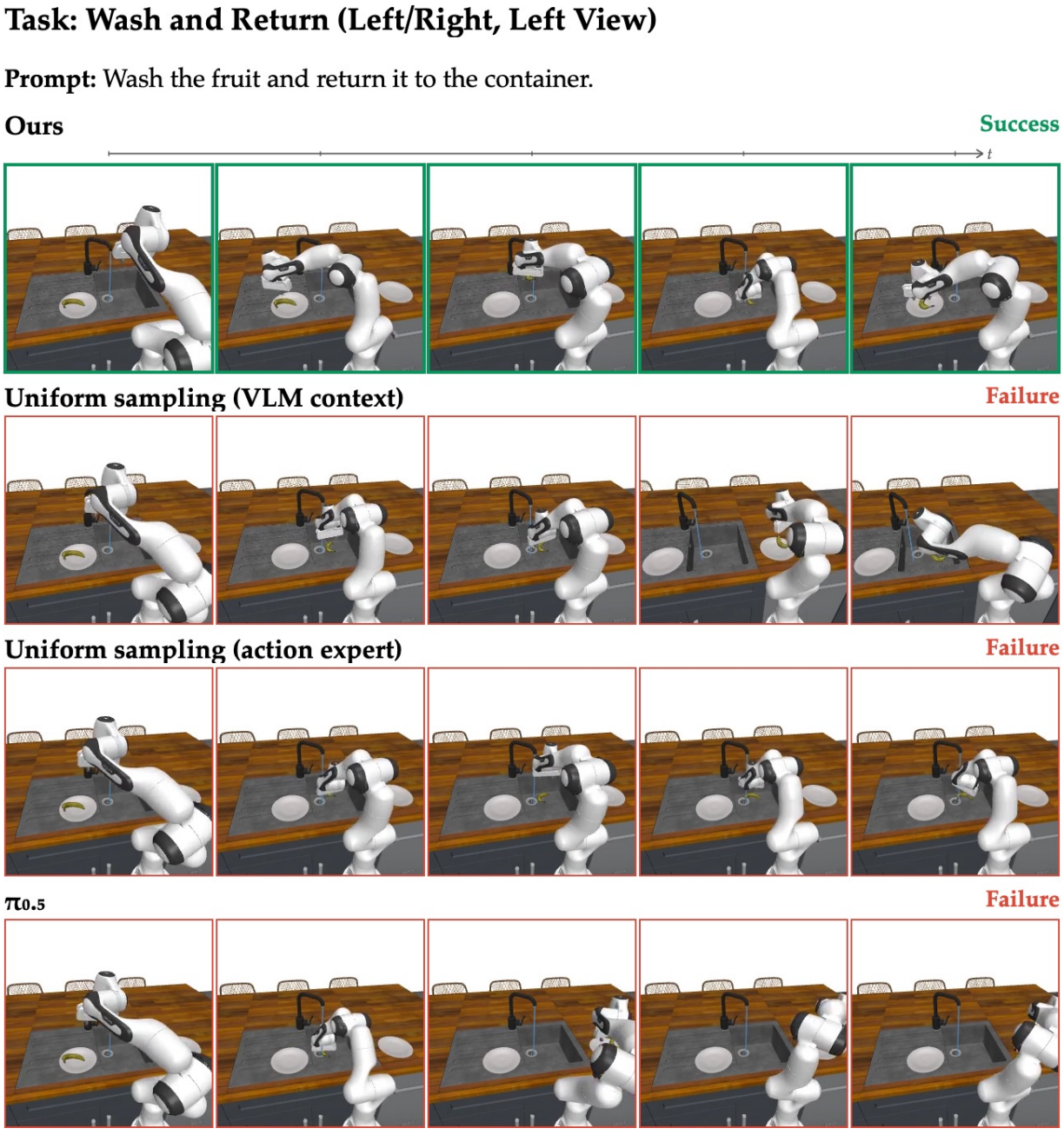


Figure 37: **RoboCasa: Wash and Return to the Original Plate.** Representative rollouts of IMPRINT, $\pi_{0.5}$, and Uniform Sampling with VLM-context or action-expert conditioning.

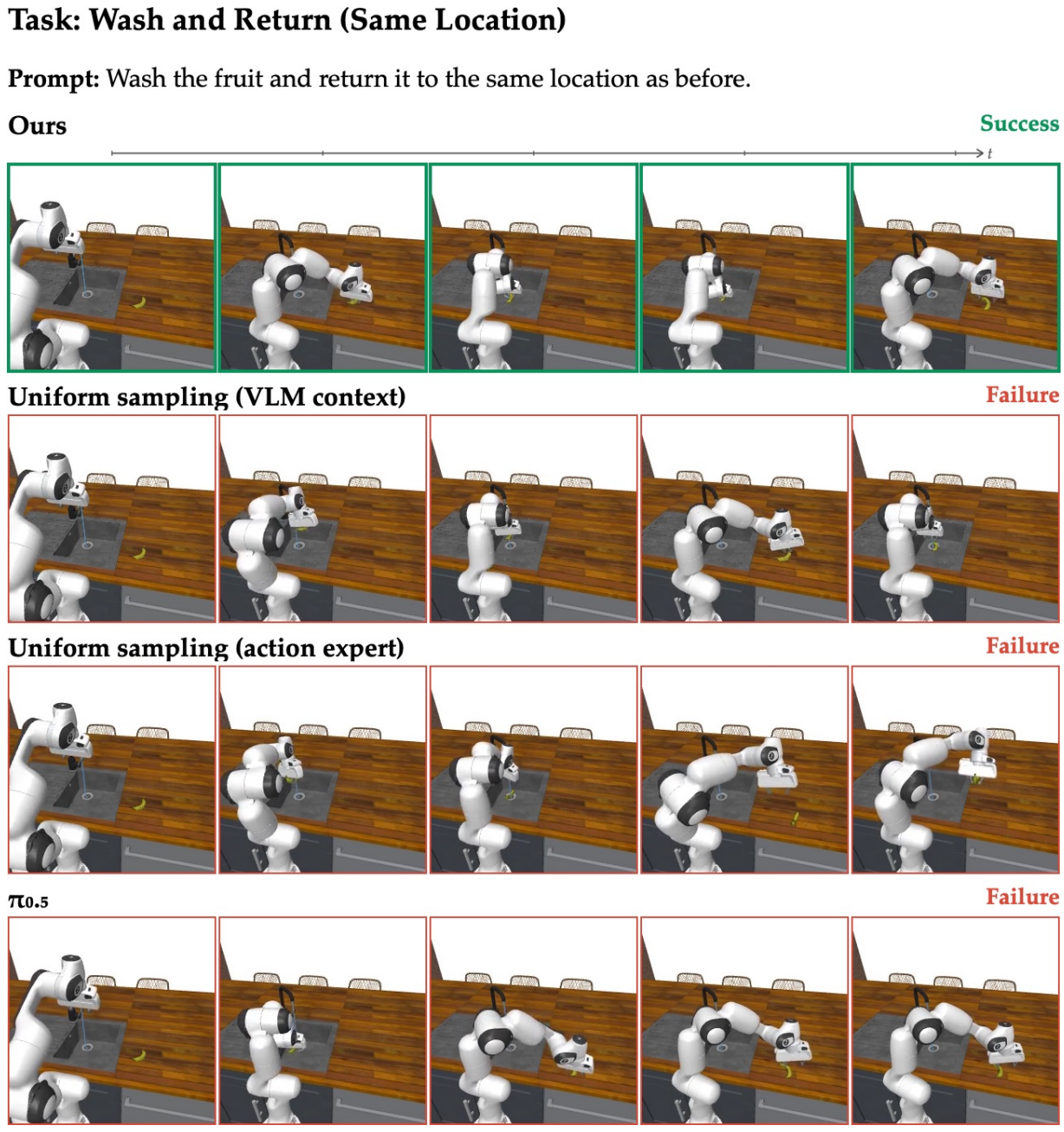


Figure 38: **RoboCasa: Wash and Return to the Same Location.** Representative rollouts of IMPRINT, $\pi_{0.5}$, and Uniform Sampling with VLM-context or action-expert conditioning.

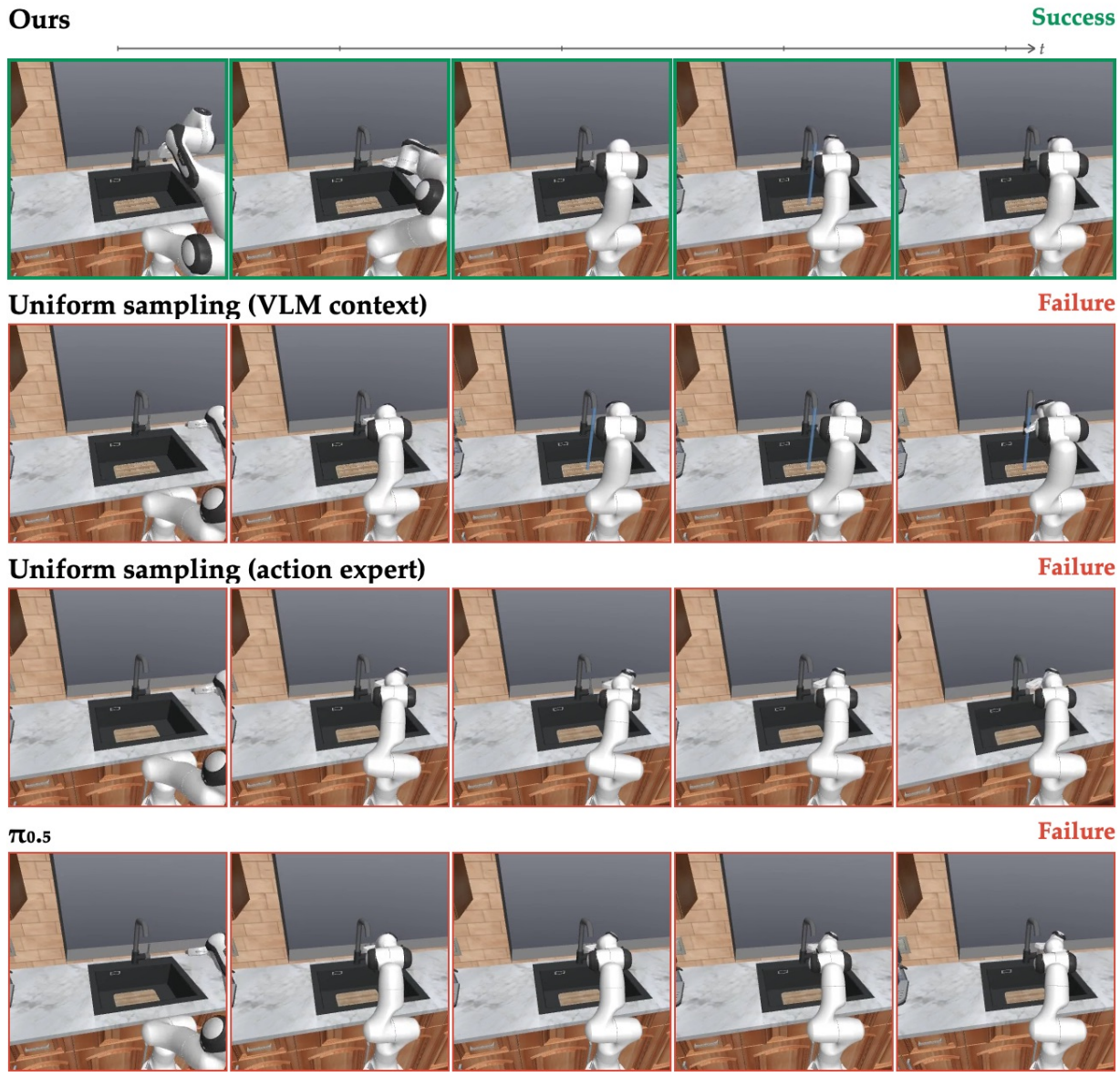


Figure 39: **RoboCasa: Rinse Cutting Board.** Representative rollouts of IMPRINT, $\pi_{0.5}$, and Uniform Sampling with VLM-context or action-expert conditioning.

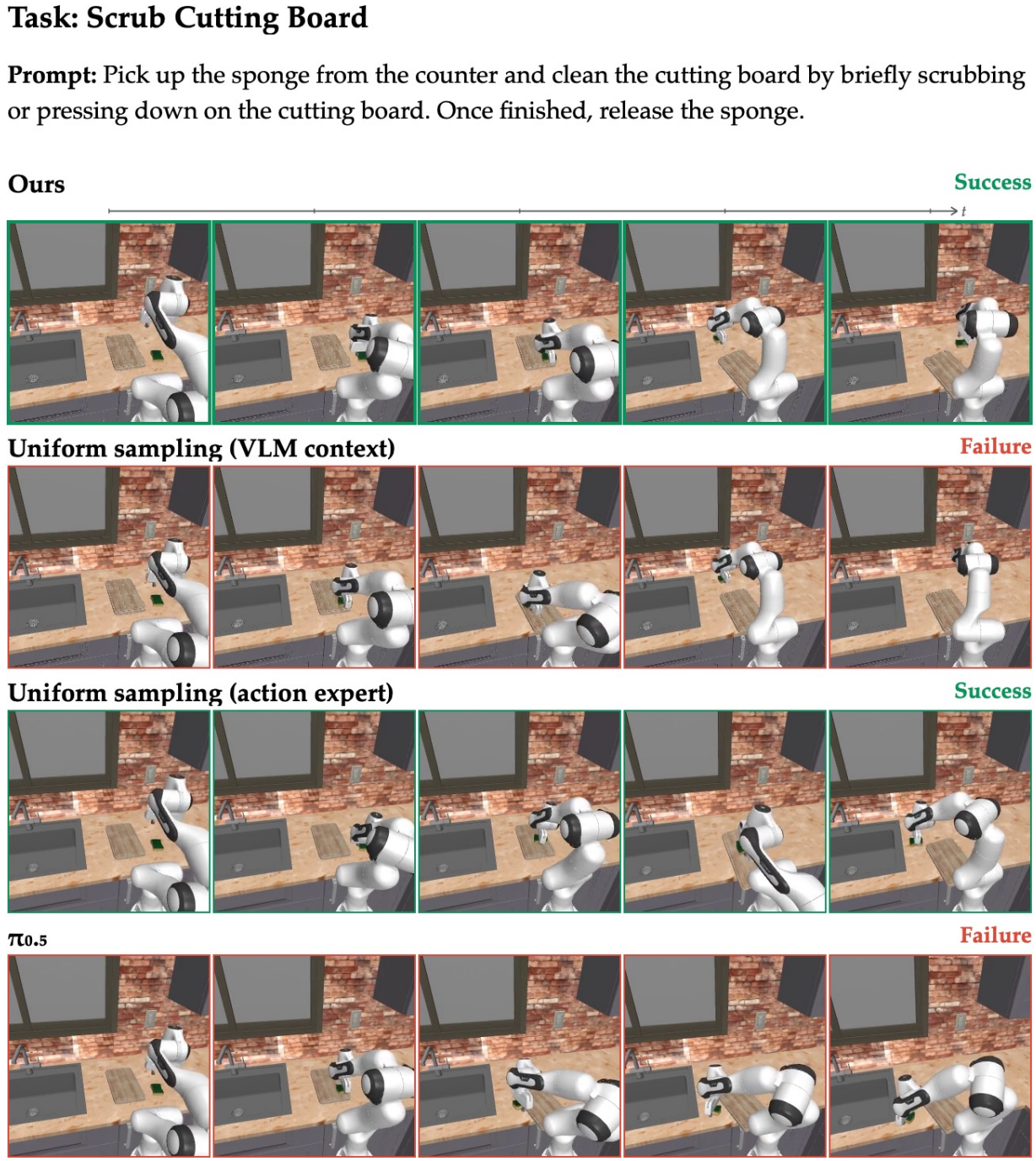


Figure 40: **RoboCasa: Scrub Cutting Board.** Representative rollouts of IMPRINT, $\pi_{0.5}$, and Uniform Sampling with VLM-context or action-expert conditioning.

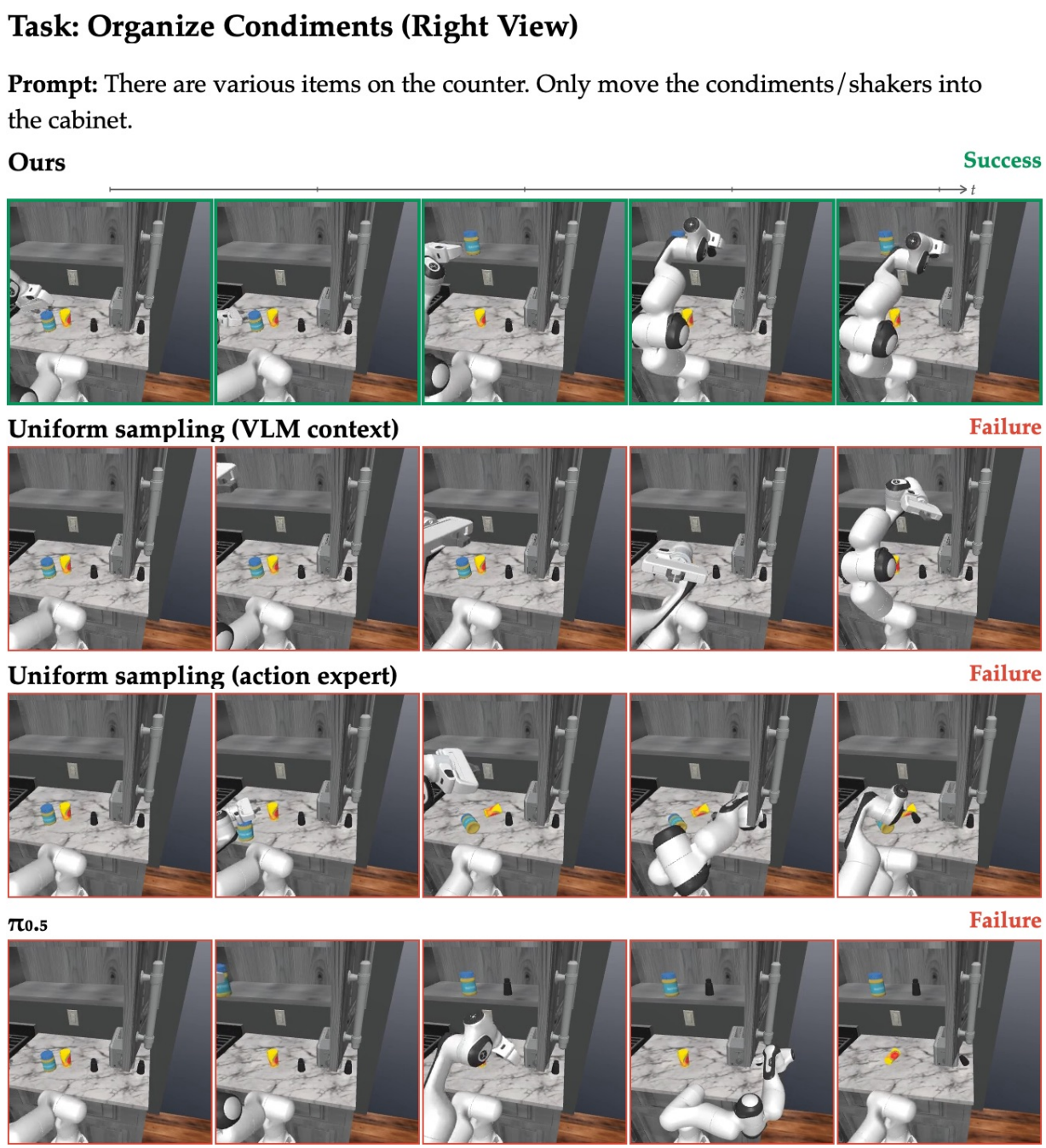


Figure 41: **RoboCasa: Organize Condiments.** Representative rollouts of IMPRINT, $\pi_{0.5}$, and Uniform Sampling with VLM-context or action-expert conditioning.

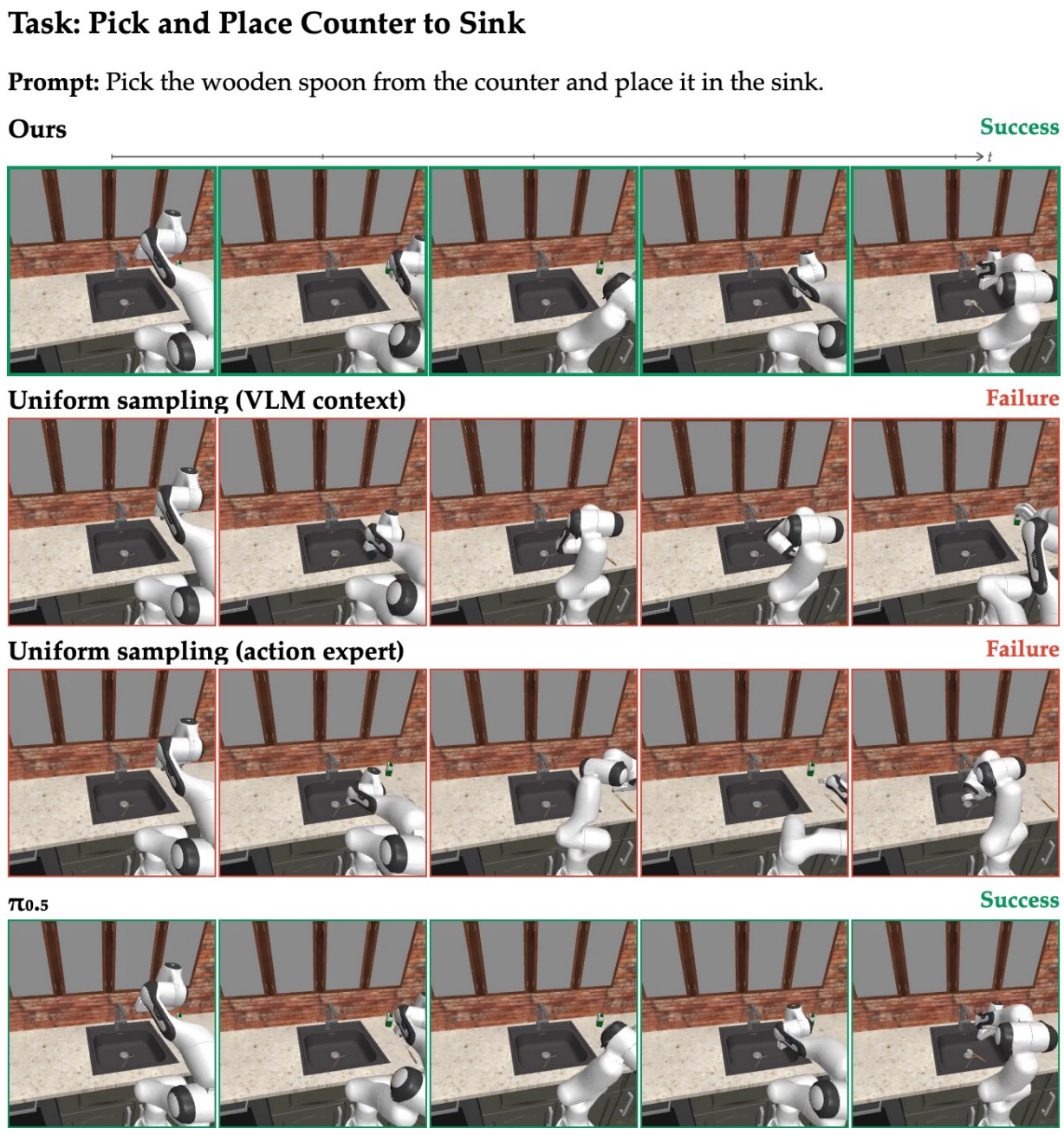


Figure 42: **RoboCasa: Pick and Place to Sink.** Representative rollouts of IMPRINT, $\pi_{0.5}$, and Uniform Sampling with VLM-context or action-expert conditioning.

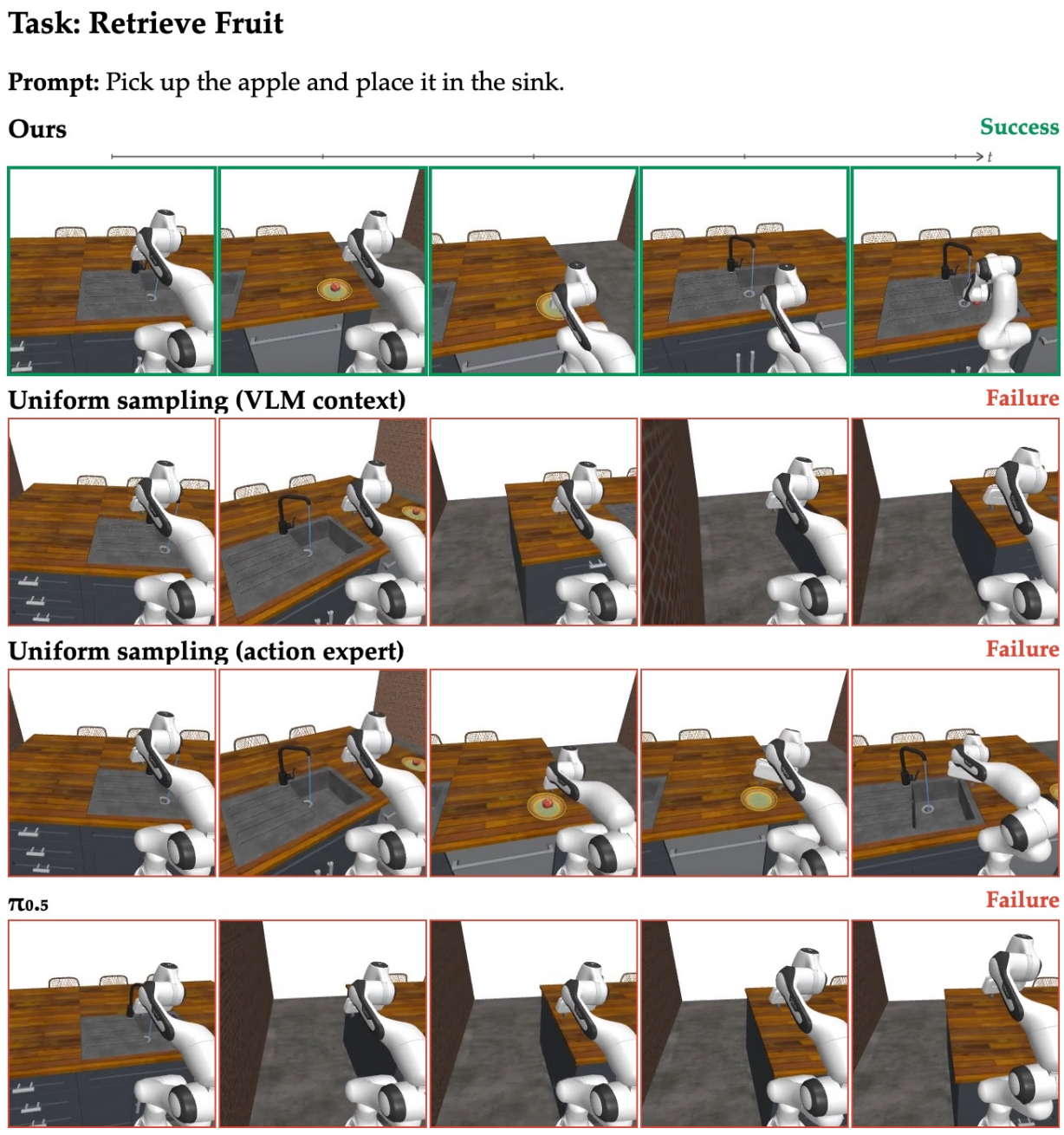


Figure 43: **RoboCasa: Retrieve Fruit.** Representative rollouts of IMPRINT, $\pi_{0.5}$, and Uniform Sampling with VLM-context or action-expert conditioning.

Retrieve Oil. As shown in Fig. 44, both IMPRINT and Uniform Sampling with action-expert conditioning grasp the correct olive oil bottle. Uniform Sampling with VLM-context conditioning approaches the wrong side of the counter, while $\pi_{0.5}$ fails to pick up the bottle.

Place Equal Ice Cubes. In Fig. 45, only IMPRINT places two ice cubes in each cup. Each of the other three methods fails to place the final required cube, leaving the task incomplete.

E.3 Real-World Analysis

We compare IMPRINT with π_0 (non-memory), HAMLET, MemER, and Uniform Sampling on the three real-world tasks, and also show IMPRINT on out-of-distribution (OOD) objects (App. B.5.3). As in simulation, the rollouts illustrate both memory errors, such as losing track of object arrangements, scanning progress, or the demonstrated folding style, and manipulation errors in otherwise correct plans. Aggregate success rates are reported in Table 12.

Dust Table. As shown in Fig. 46, IMPRINT moves both bottles off the small table, dusts it once, and returns the bottles upright in their original left/right arrangement. The same behavior holds with unseen OOD objects. HAMLET dusts the table but leaves one bottle on the workspace instead of returning it. Uniform Sampling and π_0 both return the bottles to the small table but fail the task: Uniform Sampling places them in the swapped arrangement, and π_0, which cannot recall the initial configuration, likewise does not restore it. MemER fails to pick up the last bottle showing that its high-level VLM can switch subtasks too quickly; across all trials, it succeeds in only 40% of rollouts compared with 80% for IMPRINT (Table 12).

Scan Bottle. In Fig. 47, IMPRINT picks up the bottles one at a time, scans each exactly once, and places them in a row on the other side of the table. It generalizes to the OOD objects with the same sequence. HAMLET triggers the scanner more than once and does not complete the transfer, consistent with losing track of which bottles have already been scanned. MemER scans a bottle but then returns to it instead of proceeding to the next one, matching the subtask-prediction errors noted in App. C.1.3. Uniform Sampling and π_0 both hesitate between picking and scanning and do not complete the scan-and-transfer sequence before the rollout ends. Because scanning does not visibly change the bottles, the scanning status must be inferred from the robot's past actions and proprioceptive states, which IMPRINT encodes and the visual-only baselines do not.

Fold Cloth. Fig. 48 first shows the in-context demonstration, which folds the towel in half and then folds it a second time. IMPRINT reproduces both folds and reaches a final shape close to the demonstrated one, including on an unseen towel color. The baselines instead fold in a style that differs from the example. Uniform Sampling folds the towel diagonally into a triangle, suggesting that it recalls that a fold is required but not which fold. HAMLET's and π_0's folds do not match the demonstrated sequence, and MemER completes a fold whose final shape deviates from the example. Identifying the style from the example's appearance is not sufficient; reproducing it also requires the demonstrated grasp locations and folding motion, which IMPRINT retains through its action and proprioceptive streams.

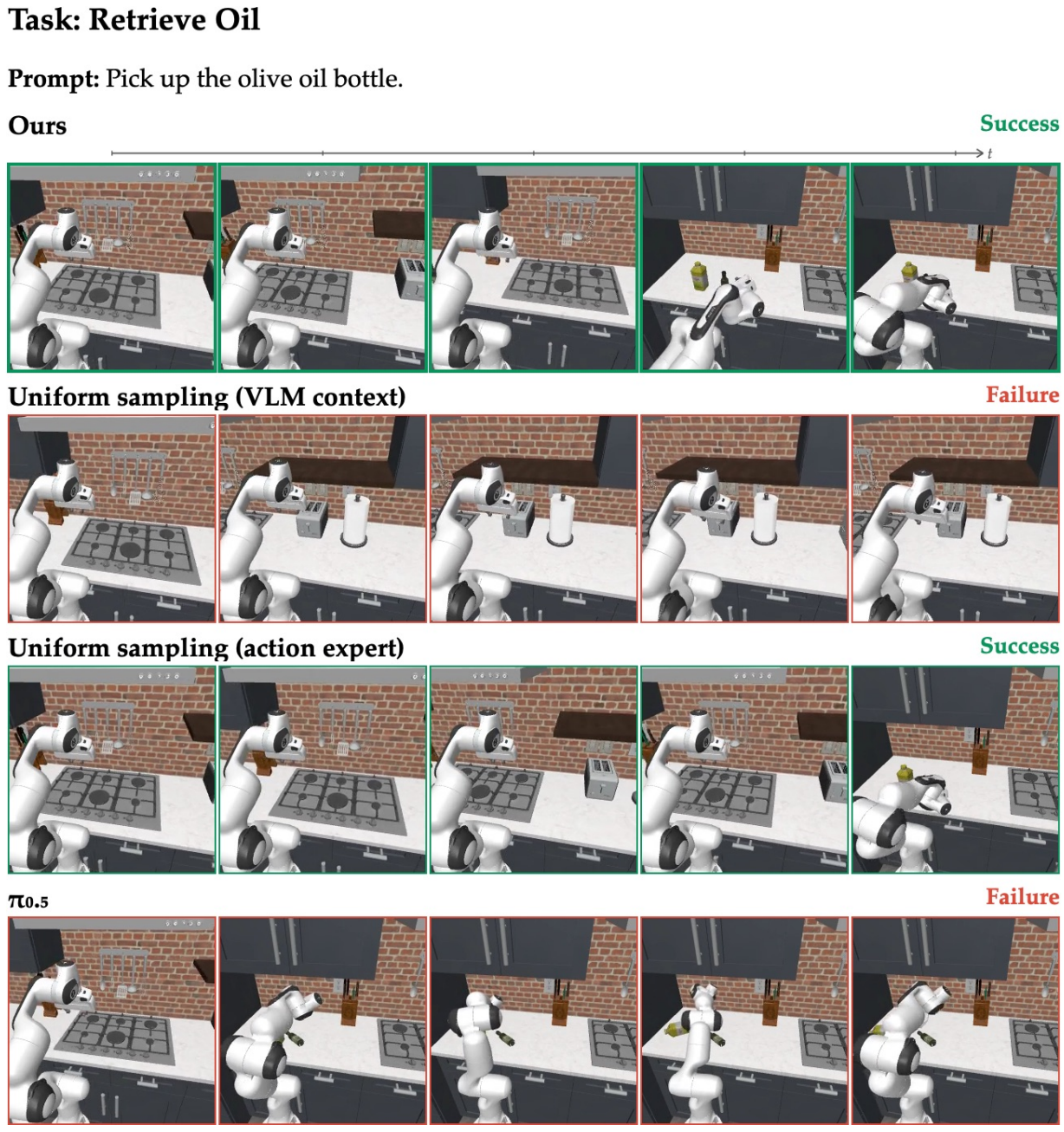


Figure 44: **RoboCasa: Retrieve Oil.** Representative rollouts of IMPRINT, $\pi_{0.5}$, and Uniform Sampling with VLM-context or action-expert conditioning.

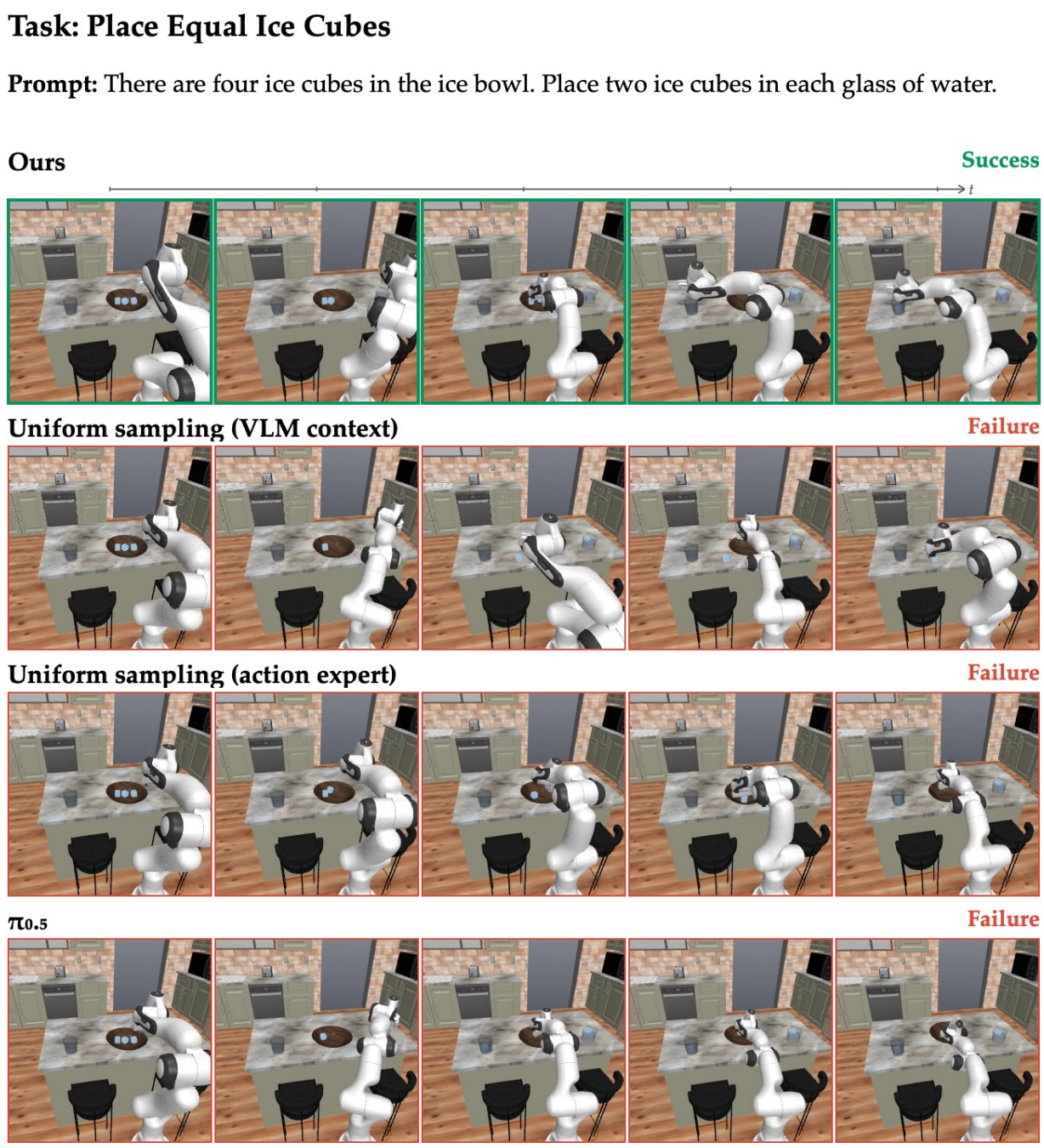


Figure 45: **RoboCasa: Place Equal Ice Cubes.** Representative rollouts of IMPRINT, $\pi_{0.5}$, and Uniform Sampling with VLM-context or action-expert conditioning.



Figure 46: **Real World: Dust Table.** Representative rollouts of IMPRINT (in-distribution and OOD objects), HAMLET, MemER, Uniform Sampling, and π_0.

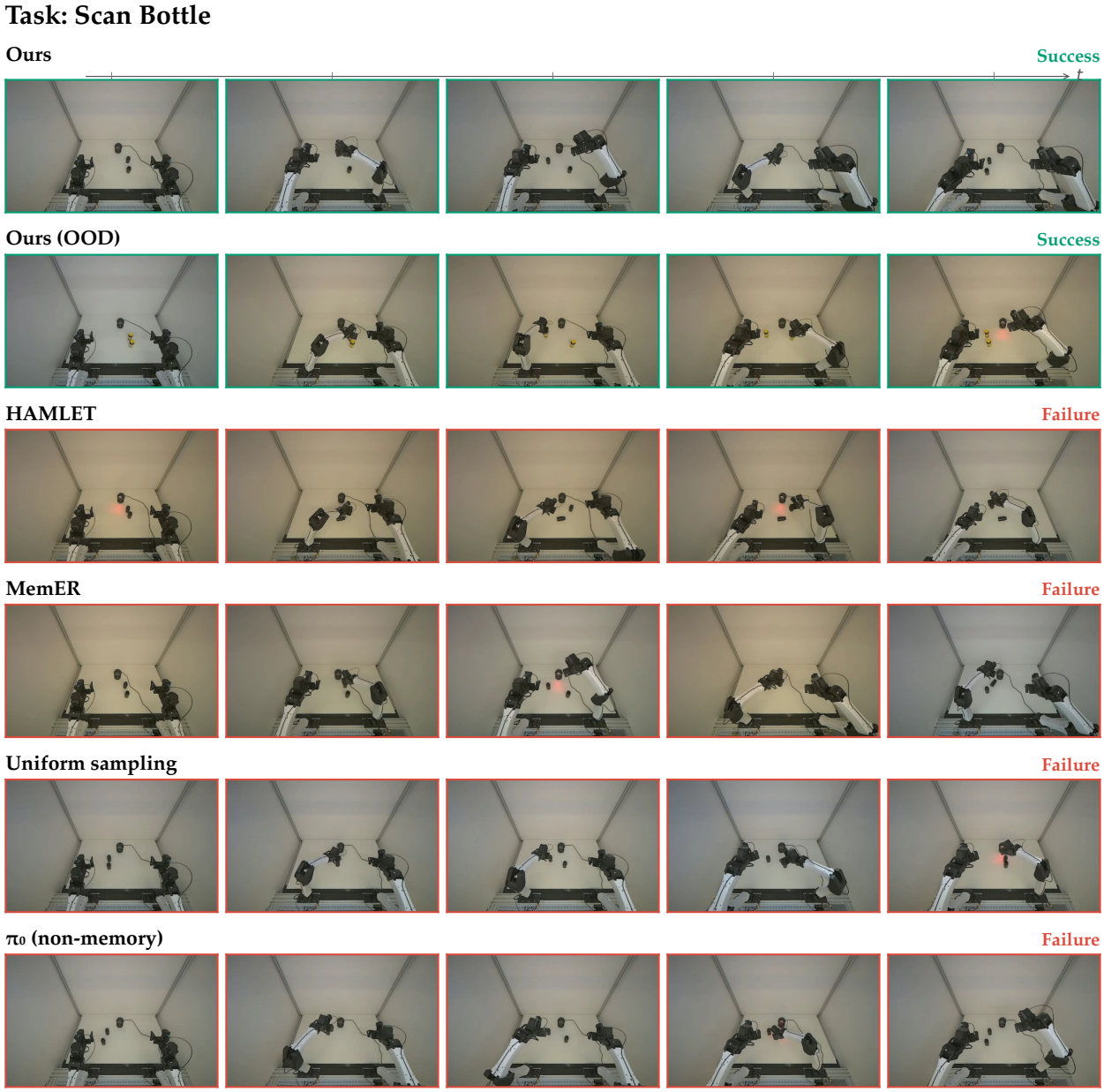


Figure 47: **Real World: Scan Bottle.** Representative rollouts of IMPRINT (in-distribution and OOD objects), HAMLET, MemER, Uniform Sampling, and π_0.

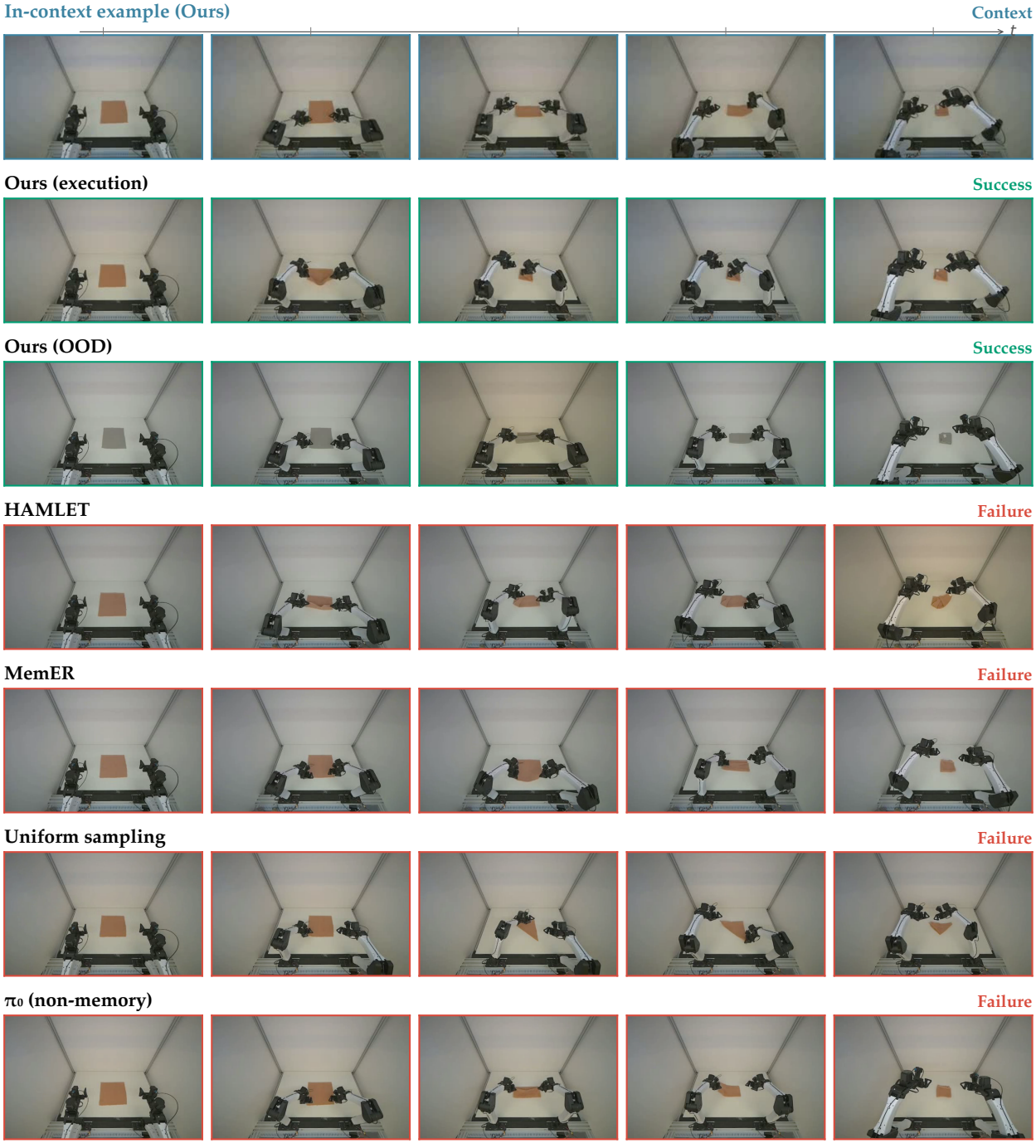


Figure 48: **Real World: Fold Cloth.** The top row shows the in-context demonstration provided to IMPRINT's memory, followed by representative rollouts of IMPRINT (in-distribution and OOD towel), HAMLET, MemER, Uniform Sampling, and π_0.