# MEMBODIED: RECURRENT ASSOCIATIVE MEMORY FOR VISION-LANGUAGE-ACTION MODELS

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

## ABSTRACT

Vision-Language-Action models provide a strong foundation for general-purpose robot control, yet a vast majority of policies do not preserve and leverage episode-level information beyond current observation. This limitation is consequential in history-dependent manipulation tasks that depend on information available only in past observations. Retaining past observations in context can aid in recovering this information, but at the significant cost of ever-growing bloated context and inference latency. We thus introduce MEMBODIED, a fixed-size episodic memory with two complementary components: an associative state that records interactions across policy calls and an episode anchor that preserves a compact representation of the initial scene as a reference. At each policy call, the model conditions action generation on the current input and the memory components, rather than directly using past observations. Across five evaluated RMBench tasks requiring memory, MEMBODIED achieves $7.81\times$ the mean success rate of a stateless policy and $2.98\times$ of vanilla recurrent memory, while outperforming the strongest memory-augmented baseline by $1.3\times$ with $10\times$ fewer added parameters. On the fully observable LIBERO-Long suite, it reached 90.6%, a 5.4% improvement over the stateless $\pi_0$ policy. These findings support MEMBODIED as a practical alternative to expanding the policy context for history-dependent manipulation.

## 1 INTRODUCTION

Vision-language-action (VLA) models are increasingly a mainstay for general robot control as they map semantic representations learned from large vision-language datasets to continuous or token-based actions. RT-2 and OpenVLA formulate actions within vision-language backbones (Brohan et al., 2023; Kim et al., 2025b); Octo (Team et al., 2024) learns a diffusion-based generalist policy from heterogeneous robot data; and $\pi_0$ couples a pretrained vision-language model to a continuous flow-matching action expert (Black et al., 2024). More recent models, such as OpenVLA-OFT (Kim et al., 2025a) and $\pi_{0.5}$ (Intelligence et al., 2025), have improved control efficiency, action generation, and open-world generalisation. Together, these developments have made VLA policies increasingly capable at complex manipulation tasks.

However, many existing VLA policies condition each policy call on the current observation and the language instruction to predict future action chunks. These approaches discard past information, limiting the policy's ability to complete memory-dependent manipulation tasks. Although action chunking promotes consistency within a predicted horizon, it does not preserve observational information across subsequent policy calls. For example, if a robot moves an object away from its initial position and must later return the object, the current observation may no longer reveal the correct destination. This is a temporal state aliasing problem, where identical or similar current observations may require different actions depending on the past observations.

Although preserving history is desirable, placing past observations directly into the policy context is costly. Retaining the complete history causes the context length, memory consumption, and inference cost to grow throughout an episode. A fixed observation window bounds these costs but may discard task-relevant evidence once it falls outside the window. Moreover, the relevant evidence may be a short event separated from the current decision by many intermediate observations. An effective memory must therefore preserve relevant information without growing in size or inference cost over the episode.



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

Recent work addresses this problem by compressing historical observations (Koo et al., 2026; Wang et al., 2026), maintaining recurrent latent representations (Cherepanov et al., 2026b; Li et al., 2026; Guo et al., 2026), or explicitly storing and retrieving historical information (Fang et al., 2025; Shi et al., 2026; Li et al., 2025; Haresh et al., 2026). Methods that retain compressed observations as history tokens can still enlarge the policy context as new information is added. Recurrent-token methods keep the recurrent state fixed in size, but rely on carried embeddings to implicitly preserve and expose relevant information through repeated transformations. Memory-bank, retrieval, and language-based approaches may instead introduce auxiliary objectives or additional processing pipelines. These considerations motivate a memory that is updated online, remains fixed in size, and retrieves information according to the policy's current state. Associative memory meets these requirements through an explicit read/write structure in which the delta rule specifies how the state is updated while the policy learns what to write, retain, and retrieve using only its action objective.

In this paper, we introduce MEMBODIED, an episodic memory for VLA control designed to preserve both interaction history and initial-scene information. Its associative memory stores interactions in layer-wise matrices through learned read and write interfaces. After each policy call, the model combines the executed action with its observed visual consequence and writes this interaction using a gated delta update. Because repeated updates may overwrite fine-grained details from early in the episode, the anchor pathway provides persistent access to a compact representation of the initial scene. At subsequent calls, information retrieved from both memory mechanisms conditions the action network. This division of roles allows past interactions and initial-scene evidence to inform action prediction while keeping the memory footprint and access cost independent of episode length.

We address three research questions in this paper: **RQ1:** How effective is MEMBODIED for memory-dependent manipulation compared with existing approaches? **RQ2:** How computationally efficient is MEMBODIED during inference? **RQ3:** Does MEMBODIED preserve performance on fully observable or Markovian manipulation where memory is not required?

Across five memory-dependent RMBench tasks, MEMBODIED achieves $7.81\times$ the mean success rate of a stateless $\pi_0$ policy and $2.98\times$ that of vanilla recurrent memory, outperforming NativeMEM's compressed-history approach by $1.30\times$. Evaluated on a second VLA backbone ($\pi_{0.5}$), MEMBODIED yields a $3.87\times$ performance increase over the stateless policy. On physical robot experiments across three tasks, MEMBODIED demonstrates an $8.0\times$ improvement in success rate. Crucially, MEMBODIED achieves these gains while reducing inference latency by $91.9\%$ relative to NativeMEM. On LIBERO, MEMBODIED achieved 95.1% mean success rate, comparable to $\pi_0$'s result of 94.2%. Notably, on the long-horizon LIBERO-Long suite, MEMBODIED reached 90.6%, exceeding the $\pi_0$ result of 85.2% by 5.4 percentage points. These results demonstrate that MEMBODIED improves history-dependent control while remaining competitive on fully observable manipulation tasks.

## 2 RELATED WORK

**Memory representations for manipulation.** HAMLET (Koo et al., 2026) and NativeMEM (Wang et al., 2026) compress observations into historical tokens. $\mu$VLA (Cherepanov et al., 2026b), ReMem-VLA (Li et al., 2026), and AVA-VLA (Xiao et al., 2026) carry recurrent latent states across control steps. SAM2Act (Fang et al., 2025), MemoryVLA (Shi et al., 2026), MAP-VLA (Li et al., 2025), and Notes-to-Self (Haresh et al., 2026) use memory banks, retrieval, or language-based records. These approaches trade off explicit access to past evidence against the cost of retaining history or introducing additional memory components. MEMBODIED instead represents episode history through an evolving associative state and an initial-scene anchor, whose readouts condition the action expert. It therefore neither appends a growing sequence of observations nor retrieves discrete past observations. We discuss these and other related work in more detail in Appendix A.

## 3 METHODOLOGY

MEMBODIED comprises two complementary memory pathways: a recurrent associative state and a fixed visual reference to the initial scene. We first describe the associative update and memory-token readout, followed by the anchor pathway, read/write schedule, and sequence training. We also evaluate attention-steering and hierarchical memory variants for comparison.

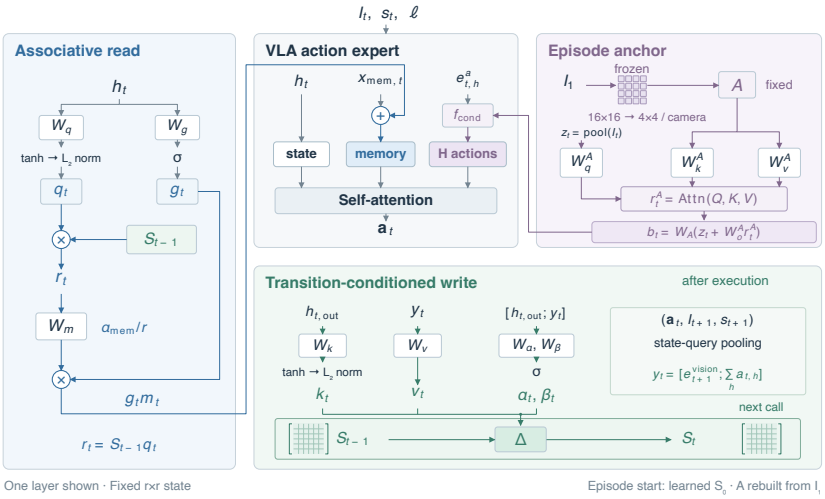



**Figure 1:** MEMBODIED architecture. An associative state records interactions, while the episode anchor preserves a reference to the initial scene.

### 3.1 RECURRENT VLA FORMULATION

We treat manipulation as a history-dependent sequential decision problem where information from prior observations might be necessary to complete the task. We denote the complete episodic memory by $\mathcal{E}_t = (\mathcal{M}_t, A)$, where $\mathcal{M}_t$ is the recurrent associative state and $A$ is the episode anchor. At interaction step $t$, the policy receives current camera observations $I_t$, robot state $s_t$, and language instruction $\ell$ to predict an action chunk $\mathbf{a}_t$ up to horizon $H$:

$$\mathbf{a}_t = [a_{t,1}, \ldots, a_{t,H}] \sim p_\theta(\cdot \mid I_t, s_t, \ell, \mathcal{E}_{t-1}).$$

The associative state $\mathcal{M}_{t-1}$ persists across policy calls and is reset to a learned initial state at the start of each episode. The anchor $A$ is constructed from the episode's first observation and remains unchanged until the episode ends.

### 3.2 ASSOCIATIVE-MEMORY MECHANISM

The memory state comprises one associative matrix $S_t^{(l)}$ for each of the $L$ action-network layers. For batch size $B$ and memory rank $r$,

$$\mathcal{M}_t = \{S_t^{(l)}\}_{l=1}^{L}, \qquad S_t^{(l)} \in \mathbb{R}^{B\times r\times r}.$$

In our experiments, we used $r = 128$. The state size remains fixed through the episode, i.e., writing new information in memory updates the existing associative matrices rather than appending observations to a growing history.

#### 3.2.1 WRITE-VALUE CONSTRUCTION

For an interaction event, we construct the value to be written from the action chunk $\mathbf{a}_t$ and its corresponding observed visual consequence, post-execution, in $I_{t+1}$. For each camera $c$, a query derived from the projected robot state $s_{t+1}$ attends to the corresponding visual patch tokens $X_{t+1}^{(c)}$:

$$e_{t+1}^{(c)} = \mathrm{softmax}\left(\frac{(W_{pq}\phi_s(s_{t+1}))(W_{pk}X_{t+1}^{(c)})^\top}{\sqrt{D}}\right) W_{pv}X_{t+1}^{(c)}$$

The camera-specific observation encodings $e_{t+1}^{(c)}$ are pooled independently and then averaged:

$$e_{t+1}^{\text{vision}} = \frac{1}{C}\sum_{c=1}^{C} e_{t+1}^{(c)}.$$

Note that, the gradients from the memory-value path are stopped at the visual patch tokens. The pooled next-observation encoding is concatenated with the aggregate of its preceding causal action chunk:

$$y_t = \left[ e_{t+1}^{\text{vision}} \; ; \; \sum_{h=1}^{H} a_{t,h} \right], \qquad v_t^{(l)} = W_v^{(l)} y_t.$$

The resulting representation $y_t$ combines the action summary and observed visual consequence, and each layer-specific projection $v_t^{(l)}$ determines the value written to that layer's associative state. This choice allows the memory value to represent a dynamic interactive transition rather than an isolated observation.

#### 3.2.2 GATED DELTA-RULE WRITE

To construct the write key at layer $l$, we transform the post-layer state representation $h_{t,\text{out}}^{(l)}$:

$$k_t^{(l)} = \text{L}_2\text{Norm}\left(\tanh(W_k^{(l)} h_{t,\text{out}}^{(l)})\right).$$

The write and retention gate values are then computed from the layer output and write value:

$$\beta_t^{(l)} = \sigma(W_\beta^{(l)}[h_{t,\text{out}}^{(l)}; y_t]), \qquad \alpha_t^{(l)} = \sigma(W_\alpha^{(l)}[h_{t,\text{out}}^{(l)}; y_t]).$$

The associative key-value state is then updated using a gated delta rule (Yang et al., 2025)

$$S_t^{(l)} = \text{Diag}(\alpha_t^{(l)}) S_{t-1}^{(l)} + \text{Diag}(\beta_t^{(l)}) \left(v_t^{(l)} - \text{Diag}(\alpha_t^{(l)}) S_{t-1}^{(l)} k_t^{(l)}\right) (k_t^{(l)})^\top.$$

The first term retains the existing state, while the second term applies a delta correction by comparing the new value with the value already associated with the write key. The retention gate $\alpha_t^{(l)}$ controls how much of the existing state remains, while the write gate $\beta_t^{(l)}$ controls the strength of the correction. The update could therefore revise an existing association without indiscriminately accumulating every value.

### 3.3 ASSOCIATIVE RETRIEVAL AND MEMORY VARIANTS

At each layer $l$, the relevant memory readout $r_t^{(l)}$ is retrieved from the respective associative matrix $S_{t-1}^{(l)}$ using a query $q_t^{(l)}$ constructed from the state token representation entering the layer, $h_{t,\text{in}}^{(l)}$:

$$q_t^{(l)} = \text{L}_2\text{Norm}\left(\tanh(W_q^{(l)} h_{t,\text{in}}^{(l)})\right), \qquad r_t^{(l)} = S_{t-1}^{(l)} q_t^{(l)}.$$

We compare the following three variants of conditioning the action network on readout $r_t^{(l)}$:

1. **Attention steering (MEMBODIED-AS):** Projections of the associative readout produce additive corrections to the attention query and output representations of the action network.
2. **Hierarchical memory (MEMBODIED-H):** An LSTM-like recurrent cell is updated from the associative matrices controlled by a learned forget gate and added back, augmenting the base gated-delta update.
3. **Memory-token readout (MEMBODIED):** The associative readout is projected to the width of the action network and added to a dedicated contextual token before self-attention.

#### 3.3.1 MEMBODIED: MEMORY-TOKEN INJECTION

In MEMBODIED, the action-network suffix is arranged as

$$[\text{state token},\ \text{memory token},\ H \text{ noisy action tokens}].$$

The memory token is a learned vector that is inserted into the suffix at every policy call. The memory token does not persist across policy calls. Instead, the associative matrices carry the recurrent state. At layer $l$, the retrieved vector is linearly transformed and scaled by $\alpha_{\text{mem}}/r$, where $\alpha_{\text{mem}}$ is the

memory scaling factor and $r$ is the memory rank. A scalar gate derived from the current state representation then controls its contribution:

$$x^{(l)}_{\mathrm{mem},t} \leftarrow x^{(l)}_{\mathrm{mem},t} + g^{(l)}_t m^{(l)}_t, \text{ where } \quad m^{(l)}_t = \frac{\alpha_{\mathrm{mem}}}{r} W^{(l)}_m r^{(l)}_t, \quad g^{(l)}_t = \sigma(W^{(l)}_g h^{(l)}_{t,\mathrm{in}})$$

The modified memory token, $x^{(l)}_{\mathrm{mem},t}$, then participates in self-attention, allowing the action tokens to use the retrieved content. Unlike attention steering, this approach makes the retrieved vector available as contextual content rather than using it only to alter existing attention computations.

### 3.4 Fixed episode-anchor memory

The associative matrices record how an episode evolves, but repeated delta-rule updates can overwrite fine-grained details of the initial scene. We therefore use a separate anchor pathway to preserve a compact, fixed visual reference derived from the episode's first observation. To construct this anchor, we take the frozen vision tokens used in the policy prefix, average-pool each camera's $16 \times 16$ patch grid into a $4 \times 4$ grid, and concatenate the pooled tokens across cameras. The resulting anchor $A \in \mathbb{R}^{N_A \times D}$ provides access to the initial scene without retaining the raw observation.

At policy call $t$, the current robot state $s_t$ attends to each camera's visual tokens $X^{(c)}_t$ using the pooling operation defined in the write-value construction. We average the camera-specific outputs to obtain $z_t$, which queries the anchor through rank-$r_A$ cross-attention:

$$c_t = z_t + W^A_o \left[ \mathrm{softmax}\left( \frac{(W^A_q z_t)(W^A_k A)^\top}{\sqrt{r_A}} \right) W^A_v A \right].$$

$c_t$ is then projected to the action-network width and repeated across the action horizon:

$$b_t = W_A c_t, \qquad b_{t,h} = b_t, \quad h = 1, \ldots, H.$$

Let $e^a_{t,h}$ denote the action representation at horizon position $h$. The anchor-conditioned representation is computed as

$$\widetilde{e}^a_{t,h} = f_{\mathrm{cond}}\big([e^a_{t,h};\, b_{t,h}]\big), \qquad h = 1, \ldots, H,$$

where $f_{\mathrm{cond}}$ denotes the action network's input-conditioning function. For our experiments, we used $r_A = 64$. The anchor remained fixed within an episode and reset at the start of a new episode.

The anchor does not append the first image or its complete token sequence to the policy prefix. Instead, the current observation selectively retrieves a compact spatial reference through cross-attention, leaving the prefix length, token positions, and attention-mask structure unchanged.

### 3.5 Causal coordination of reading and writing

At step $t$, the policy first reads $\mathcal{M}_{t-1}$ and generates $\mathbf{a}_t$. After the action chunk is executed, the environment supplies the resultant visual observation, $I_{t+1}$. The model then associates the hidden state and action chunk from step $t$ with $I_{t+1}$ and writes the association to $\mathcal{M}_t$:

$$\mathcal{M}_{t-1} \xrightarrow{\text{read with } (I_t, s_t, \ell)} \mathbf{a}_t \xrightarrow{\text{execute}} I_{t+1} \xrightarrow{\text{write}} \mathcal{M}_t.$$

This delayed-write schedule allows the stored value to include the observed consequence of an action without exposing that future observation to the action that caused it. During inference, the policy caches the preceding layer states and the sampled action chunk from the final denoising step, and performs the write when the next observation arrives. Thus, training and inference use the same causal interpretation of an interaction event.

### 3.6 Sequence training

The delayed write makes sequence-level training necessary. An action loss at a later policy call must propagate through memory operations performed at earlier calls to train the memory parameters. Each training sample comprises a sequence of $N$ observation-action pairs. Consecutive pairs

are separated by one action horizon of $H$ environment steps, approximating successive policy calls during execution. Image encoding and policy-input construction are parallelised across the batch and sequence dimensions, while the memory state is propagated sequentially through the sequence. Gradients pass through the full sequence, allowing later action losses to optimise earlier memory operations. The memory parameters are thus learned jointly through the policy's native action objective, without a separate memory-prediction loss.

# 4 Experimental Setup

We evaluate MemBodied on memory-dependent manipulation using RMBench (Chen et al., 2026) with the $\pi_0$ and $\pi_{0.5}$ backbones, and on three real-robot tasks. We also use LIBERO (Liu et al., 2023) to assess whether MemBodied preserves performance on fully observable manipulation tasks.

## 4.1 Benchmarks and Evaluation Protocol

RMBench (Chen et al., 2026) is a bimanual simulation benchmark that organises tasks by memory complexity. Its $M(1)$-category of tasks require retention of task-relevant observations from a single past event, whereas the $M(n)$ tasks require information from multiple past events. Our five-task evaluation covered three $M(1)$ tasks: `put_back_block`, `rearrange_blocks`, and `swap_blocks`; and two $M(n)$ tasks: `battery_try` and `block_ranking_try`. We conducted 50 rollouts per task and report each task's success rate and the mean across the five tasks.

As a complementary general manipulation evaluation, we use the four standard LIBERO suites: LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long. Each suite contains ten tasks, and we conduct 50 rollouts per task, yielding 500 rollouts per suite and 2,000 rollouts per checkpoint. We report the mean success rate for each suite and the unweighted mean across all four suites. On both benchmarks, memory is reset at the start of each rollout and persists only for the duration of that episode.

## 4.2 Baselines

Our RMBench-centered comparison includes $\pi_0$-based baselines, published benchmark results, and variants of our method. The locally trained policies share the same $\pi_0$ backbone, LoRA adaptation settings, and number of optimiser steps. These controls align the adaptation recipe and training-step budget, but do not imply identical architectures or compute costs. The full-fine-tuning LIBERO comparisons are reported separately.

**$\pi_0$-Stateless** takes only the current observation, robot state, and instruction as input, essentially a memory-free baseline. **$\pi_0$-FrameStack** additionally receives a sequence of past observations and serves as an in-context history baseline. **$\pi_0$-Hint** augments the stateless policy with simulator-derived textual hints on task progression at inference time, without any training on those hints; it is therefore a privileged-information reference. **$\pi_0$-Vanilla Recurrent Memory** adapts the associative memory mechanism of Delta-Mem (Lei et al., 2026), while **$\pi_0$-$\mu$-VLA** adapts the recurrent memory-token mechanism of $\mu$VLA (Cherepanov et al., 2026b).

We report the Diffusion Policy, ACT, $\pi_{0.5}$, and X-VLA results from Chen et al. (2026), and use the $\pi_{0.5}$ scores as a full-fine-tuned $\pi_{0.5}$ baseline. With reference to our method, we compare the attention-steering (MemBodied-AS), hierarchical (MemBodied-H), and memory-token (MemBodied) variants of the associative mechanism. Finally, for comparison with Native-MEM (Wang et al., 2026), we integrate its video-history encoding into our setup and evaluate it both independently and in combination with MemBodied. The full implementation and evaluation details are provided in Appendix B and C.

# 5 Results

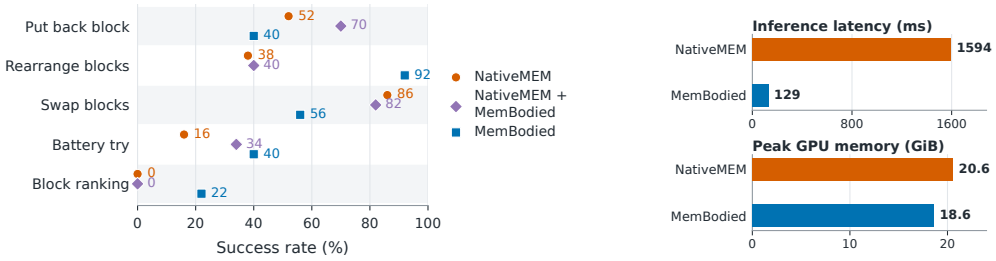
## 5.1 Performance on memory-dependent manipulation

**Recurrent memory and its formulation both affect performance.** To assess the effectiveness of MemBodied for memory-dependent manipulation (RQ1), we compare it with baselines and Mem-

Table 1: Success rates (%) on the five evaluated RMBench tasks. Published policy values are taken from Chen et al. (2026). The bold figures indicate the best result for each task/column.

| Model | Put back block | Rearrange blocks | Swap blocks | Battery try | Block ranking | *Mean* |
|---|---|---|---|---|---|---|
| *Published policies (Chen et al., 2026)* | | | | | | |
| Diffusion Policy | 0.0 | 0.0 | 11.0 | 10.0 | 10.0 | 6.2 |
| ACT | 0.0 | 29.0 | 2.0 | 19.0 | 0.0 | 10.0 |
| X-VLA | 18.0 | 13.0 | 16.0 | 26.0 | 1.0 | 14.8 |
| $\pi_{0.5}$ | 11.0 | 13.0 | 16.0 | 16.0 | 6.0 | 12.4 |
| *Baselines in our setup ($\pi_0$ backbone)* | | | | | | |
| $\pi_0$-Stateless | 6.0 | 2.0 | 8.0 | 16.0 | 0.0 | 6.4 |
| $\pi_0$-Hint | 10.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 |
| $\pi_0$-$\mu$-VLA | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| $\pi_0$-FrameStack | 14.0 | 30.0 | 0.0 | 4.0 | 26.0 | 14.8 |
| $\pi_0$-Vanilla Recurrent Memory | 18.0 | 22.0 | 14.0 | 22.0 | 8.0 | 16.8 |
| MEMBODIED-AS | 32.0 | 26.0 | 12.0 | 20.0 | 14.0 | 20.8 |
| MEMBODIED-H | 16.0 | 50.0 | 14.0 | 28.0 | 8.0 | 23.2 |
| MEMBODIED w/o anchor memory | 34.0 | 76.0 | 16.0 | 30.0 | **32.0** | 37.6 |
| MEMBODIED | **40.0** | **92.0** | **56.0** | **40.0** | 22.0 | **50.0** |

BODIED variants across five RMBench tasks. As shown in Table 1, MEMBODIED reaches 50.0% mean success, exceeding $\pi_0$-Stateless by 43.6 percentage points and $\pi_0$-FrameStack by 35.2 points; its advantage over the stateless policy is positive across all five tasks. These comparisons support the value of maintaining a recurrent memory across policy calls rather than ignoring history or presenting a bounded history directly in context. MEMBODIED also surpasses $\pi_0$-Vanilla Recurrent Memory by 33.2 points; as both policies maintain a recurrent state, this comparison indicates the merit of our proposed memory formulation beyond recurrence alone.

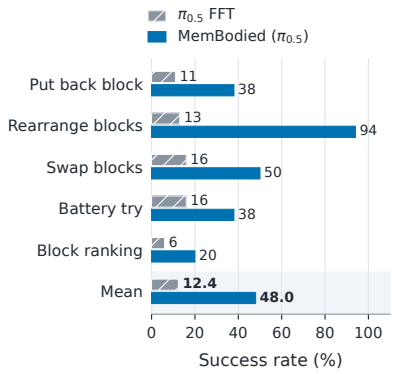



**(a)** Success on each RMBench task. **(b)** Inference Efficiency Comparison.

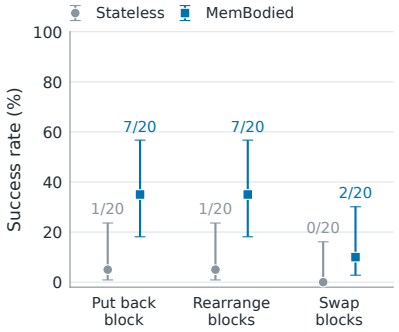
**Figure 2:** NativeMEM and MEMBODIED in the same evaluation setup. We evaluate NativeMEM's video-history encoding alone and combined with MEMBODIED, alongside standalone MEMBODIED. Standalone MEMBODIED exceeds NativeMEM by 11.6 percentage points on average.

**Comparison with compressed history methods.** Following Fig. 2, even with this more granular access to history, NativeMEM reaches 38.4% mean success, compared with 50.0% for standalone MEMBODIED. Adding MEMBODIED to NativeMEM improves its mean to 45.2%, but remains 4.8% below standalone MEMBODIED. The gain from adding MEMBODIED to NativeMEM could reflect more focused memory retrieval by MEMBODIED. However, NativeMEM's history tokens could also draw attention away from the information retrieved by MEMBODIED, potentially explaining why the combined model performs worse than standalone MEMBODIED. Exploring how to combine explicit video history with associative memory more effectively is a direction for future work.

However, as seen in Fig. 2a, no single method dominates at the task level. NativeMEM is stronger than standalone MEMBODIED on Put Back Block and Swap Blocks, whereas MEMBODIED is stronger on Rearrange Blocks, Battery Try, and Block Ranking. The combined model obtains the highest Put Back Block rate, but does not surpass the strongest standalone methods on the other four tasks. These results favour MEMBODIED in the five-task mean success rate while showing that compressed history and associative memory have complementary, task-dependent strengths.




**(a)** $\pi_{0.5}$ FFT baseline and MEMBODIED.




**(b)** Real robot Evaluation. Whiskers are marginal 95% Wilson binomial confidence intervals.

**Figure 3:** Additional backbone and real-robot memory tasks.

**The gains transfer to $\pi_{0.5}$ backbone.** With $\pi_{0.5}$ backbone (Fig. 3a), MEMBODIED reaches 48.0% mean success as compared to 12.4% for the $\pi_{0.5}$ baseline, improving on all five evaluated tasks. The largest gain is achieved on Rearrange Blocks, from 13.0% to 94.0%; Put Back Block improves from 11.0% to 38.0%.

**MEMBODIED improves real-robot performance.** As shown in Fig. 3b, MEMBODIED increases mean success across three real-robot tasks from 3.33% to 26.67%, with improvements on every task.

### 5.2 INFERENCE EFFICIENCY

As shown in Fig. 2b, MEMBODIED achieves 91.9% lower inference latency while using 9.5% less peak GPU memory, as compared to NativeMEM. MEMBODIED also introduces substantially fewer 40M parameters (only 1.26% of $\pi_0$), roughly one-tenth of NativeMEM's 415M (12.81% of $\pi_0$). During inference, NativeMEM encodes the incoming observations with its video-history encoder and adds the resulting tokens to the VLA context, introducing an additional encoding step and a growing context size. In contrast, MEMBODIED maintains a fixed-size recurrent state that is read during action generation and updated at the start of a new policy call. Together, these results show that MEMBODIED achieves higher mean success than NativeMEM with substantially lower inference latency, parameter overhead, and memory footprint.

### 5.3 PERFORMANCE ON FULLY-OBSERVABLE MANIPULATION

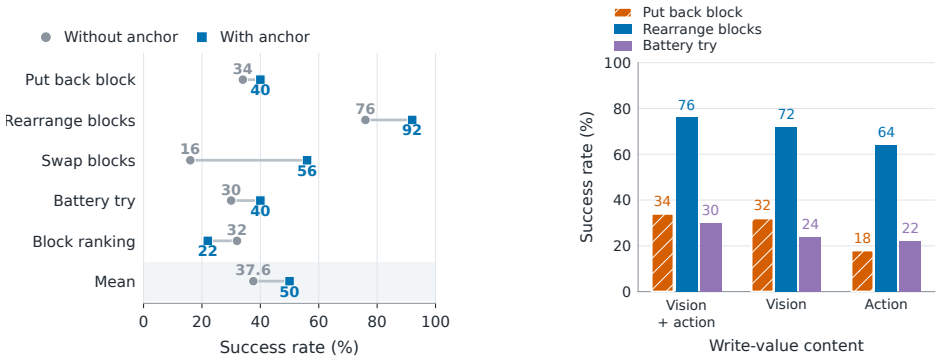
**The largest improvement is skewed toward long-horizon LIBERO tasks.** LIBERO provides a complementary setting for RQ3 by testing MEMBODIED on fully observable tasks where episodic recall is not required to complete the task. Table 2 shows that MEMBODIED reaches 90.6% success on LIBERO-Long, exceeding $\pi_0$'s score of 85.2% by 5.4 percentage points. In contrast, Goal is unchanged, while Spatial and Object decrease by 0.6 and 1.0 points, respectively. The four-suite mean remains comparable at 95.1% against $\pi_0$'s 94.2%. The skewness of the improvement toward LIBERO-Long supports the view that MEMBODIED is particularly useful for improving task performance on longer fully-observable manipulation, without degrading aggregate performance.

Table 2: **LIBERO success rates (%).**

| Method | Spatial | Object | Goal | Long | Mean |
|---|---|---|---|---|---|
| Diffusion Policy (Chi et al., 2025) | 78.3 | 92.5 | 68.3 | 50.5 | 72.4 |
| Octo (Team et al., 2024) | 78.9 | 85.7 | 84.6 | 51.1 | 75.1 |
| OpenVLA (Kim et al., 2025b) | 84.7 | 88.4 | 79.2 | 53.7 | 75.9 |
| SpatialVLA (Qu et al., 2025) | 88.2 | 89.9 | 78.6 | 55.5 | 78.1 |
| UniACT (Zheng et al., 2025) | 77.0 | 87.0 | 77.0 | 70.0 | 76.8 |
| $\pi_0$ (Black et al., 2024) | **96.8** | **98.8** | **95.8** | 85.2 | 94.2 |
| $\pi_0$-FAST (Pertsch et al., 2025) | 96.4 | 96.8 | 88.6 | 60.2 | 85.5 |
| MEMBODIED | 96.2 | 97.8 | **95.8** | **90.6** | **95.1** |

## 6 ANALYSES

**Memory architecture ablation.** Figure 4a and Table 1 compare the associative-state variants and the contribution of the episode anchor pathway. Among the anchor-free variants, memory-token




**(a)** Fixed episode anchor. **(b)** Write-value composition.

**Figure 4:** Memory design ablations on RMBench.

injection reaches 37.6% mean success, compared with 23.2% for hierarchical memory and 20.8% for attention steering. This indicates that feeding the retrieved vector as contextual content is more effective, in our setting, than the two alternative memory designs. The combined two-pathway memory reaches 50.0%, compared with 37.6% when the anchor pathway is removed. Relative to the anchor-free variant, the MEMBODIEDimproves on four out of five tasks, with the largest gain on Swap Blocks, from 16.0% to 56.0%. However, the anchor reduces performance on Block Ranking from 32.0% to 22.0%; this task depends more on tracking evolving task progress than recalling the initial scene, making the fixed first-frame reference potentially less useful. Overall, both the memory-token design and episode anchor contribute to the complete memory formulation, although the anchor's effect is task-dependent.

**Vision and action play complementary roles through memory.** Across the three evaluation tasks, Fig. 4b shows that combining the subsequent visual observation with the action summary achieves 46.7% mean success, compared with 42.7% for vision-only and 34.7% for action-only memory. On Put Back Block and Rearrange Blocks, visual information accounts for most of the measured performance, with the action summary adding 2 and 4 percentage points. On Battery Try, the combined value reaches 30.0%, compared with 24.0% for vision alone and 22.0% for action alone; removing either modality therefore reduces success by 6 or 8 points. The combined representation is the strongest on every task, indicating the complementarity of these two channels.

**Qualitative analysis on the impact of memory.** Manual annotation of 50 seed-matched rollout pairs per task shows that MEMBODIED and $\pi_0$-Stateless behave similarly when the current observation is sufficient, but diverge at decisions requiring past information. For example, on Put Back Block, MEMBODIED reduces wrong-pad returns from 37/50 to 13/50, although both policies complete the preceding block movement and button press in every rollout (see more details in Appendix E). Across tasks, MEMBODIED more often selects the episode-specific target, tracks task progress, and uses earlier outcomes to guide subsequent attempts. Its remaining failures largely involve motor execution or longer-term tracking. These observations localise MEMBODIED's gains to decisions that require information no longer visible in the current observation.

## 7 CONCLUSION

In this work, we introduced MEMBODIED, a fixed-capacity episodic memory that combines an associative memory state with an initial-scene anchor without expanding the policy context. Across five memory-dependent RMBench tasks, it achieved 50.0% mean success, outperforming stateless, frame-history, vanilla recurrent, and compressed-history baselines. The improvements also extended to the $\pi_{0.5}$ backbone and three real-robot tasks. Importantly, the gains in memory-dependent manipulation do not come at the expense of inference efficiency. Compared with NativeMEM, MEMBODIED pairs higher mean success with substantially lower latency and memory footprint. MEMBODIED also preserves general manipulation ability, maintaining comparable mean success on LIBERO while improving on the long-horizon suite. Together, these results support MEMBODIED as a practical alternative to expanding observation histories in VLA control.

### AI USE STATEMENT

In this work, we used generative AI tools to provide feedback on research methodology and experiments, assist with method implementation, create or modify scientific figures, summarise existing literature, and draft or edit portions of the manuscript for clarity. We did not use these tools to develop theoretical models, formulate or prove mathematical claims, propose or refine hypotheses, clean or reformat datasets, generate synthetic data, or translate the manuscript. We manually checked LLM-generated code for correctness, verified literature summaries against the cited sources, and reviewed all AI-generated text and figures. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

## REFERENCES

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. $\pi_0$: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

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*, 2026.

Egor Cherepanov, Nikita Kachaev, Alexey Kovalev, and Aleksandr I. Panov. Memory, benchmark & robots: A benchmark for solving complex tasks with reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=9cLPurIZMj.

Egor Cherepanov, Nikita Kachaev, Daniil Zelezetsky, Aydar Bulatov, Artem Pshenitsyn, Yuri Kuratov, Alexey Skrynnik, Aleksandr I Panov, and Alexey K Kovalev. $\mu$ vla: On recurrent memory for partially observable manipulation in vla models. *arXiv preprint arXiv:2606.12497*, 2026b.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Haoquan Fang, Markus Grotz, Wilbert Pumacay, Yi Ru Wang, Dieter Fox, Ranjay Krishna, and Jiafei Duan. Sam2act: Integrating visual foundation model with a memory architecture for robotic manipulation. *arXiv preprint arXiv:2501.18564*, 2025.

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

Sanjay Haresh, Daniel Dijkman, Apratim Bhattacharyya, and Roland Memisevic. to-self: Scratchpad augmented vlas for memory dependent manipulation tasks. *arXiv preprint arXiv:2602.21013*, 2026.

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.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025a.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 06–09 Nov 2025b. URL https://proceedings.mlr.press/v270/kim25c.html.

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, pp. 101537–101558, 2026.

Jingdi Lei, Di Zhang, Junxian Li, Weida Wang, Kaixuan Fan, Xiang Liu, Qihan Liu, Xiaoteng Ma, Baian Chen, and Soujanya Poria. $\delta$-mem: Efficient online memory for large language models. *arXiv preprint arXiv:2605.12357*, 2026.

Hang Li, Fengyi Shen, Dong Chen, Liudi Yang, Xudong Wang, Jinkui Shi, Zhenshan Bing, Ziyuan Liu, and Alois Knoll. Remem-vla: Empowering vision-language-action model with memory via dual-level recurrent queries. *arXiv preprint arXiv:2603.12942*, 2026.

Runhao Li, Wenkai Guo, Zhenyu Wu, Changyuan Wang, Haoyuan Deng, Zhenyu Weng, Yap-Peng Tan, and Ziwei Wang. Map-vla: Memory-augmented prompting for vision-language-action model in robotic manipulation. *arXiv preprint arXiv:2511.09516*, 2025.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Yan Ding, Zhigang Wang, JiaYuan Gu, Bin Zhao, Dong Wang, et al. Spatialvla: Exploring spatial representations for visual-language-action model. *arXiv preprint arXiv:2501.15830*, 2025.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International conference on machine learning*, pp. 9355–9366. PMLR, 2021.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992. doi: 10.1162/neco.1992.4.1.131.

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, pp. 18567–18602, 2026.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

Ziye Wang, Modi Shi, Chaojun Ni, Jiazhi Yang, Mengdi Li, Zhizhong Su, Tianwei Lin, and Hongyang Li. Nativemem: Native memory compression for long-horizon robotic manipulation. *arXiv preprint arXiv:2607.06678*, 2026.

Lei Xiao, Jifeng Li, Juntao Gao, Feiyang Ye, Yan Jin, Jingjing Qian, Jing Zhang, Yong Wu, and Xiaoyuan Yu. Ava-vla: Improving vision-language-action models with active visual attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13453–13463, 2026.

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

Jinliang Zheng, Jianxiong Li, Dongxiu Liu, Yinan Zheng, Zhihao Wang, Zhonghong Ou, Yu Liu, Jingjing Liu, Ya-Qin Zhang, and Xianyuan Zhan. Universal actions for enhanced embodied foundation models. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22508–22519. IEEE, 2025.

# A Extended Related Work

This section expands the review in Section 2, providing additional context on the broader literature surrounding generalist VLA policies, memory-augmented manipulation, associative memory, and memory-focused evaluation.

## A.1 Generalist VLA policies

Vision-language-action models transfer semantic representations from pretrained vision-language models into robot control. RT-1 consumes a bounded image history and compresses visual tokens before causal processing (Brohan et al., 2022). RT-2 and OpenVLA represent robot actions as tokens within vision-language backbones (Brohan et al., 2023; Kim et al., 2025b). Octo instead trains a diffusion-based generalist policy from heterogeneous robot trajectories (Team et al., 2024). $\pi_0$ combines a pretrained PaliGemma vision-language model with a flow-matching action expert for continuous action chunk prediction (Black et al., 2024). OpenVLA-OFT shows that continuous action representations, parallel decoding, and action chunking can improve control speed and success, while $\pi_{0.5}$ combines low-level control with semantic prediction and heterogeneous co-training for open-world tasks (Kim et al., 2025a; Intelligence et al., 2025).

Across these architectures, temporal context is limited to a bounded observation history or an action chunk. Such local context can preserve recently visible evidence, but it does not provide a persistent episodic state for information that must be retained beyond the input window or selectively carried across an episode.

## A.2 Memory representations for manipulation

Recent memory-aware policies can be distinguished by the representation they preserve. **Compressed-history methods** retain selected observations as tokens. HAMLET (Koo et al., 2026) encodes observations into moment tokens and aggregates them through a lightweight history module. NativeMEM (Wang et al., 2026) reuses the VLA vision encoder to compress each historical camera frame into a native memory token. These methods preserve explicit temporal evidence more efficiently than raw frame histories, although their context can still grow with the number of retained observations.

**Fixed-size recurrent methods** compress history into a bounded latent state. $\mu$VLA carries learnable memory tokens across OpenVLA-OFT control steps and trains them with truncated backpropagation through time (Cherepanov et al., 2026b). ReMem-VLA propagates separate frame and chunk level recurrent queries (Li et al., 2026), while AVA-VLA uses recurrent state to modulate current visual features (Xiao et al., 2026).

**Memory-bank and retrieval methods** provide more structured storage. SAM2Act (Fang et al., 2025) integrates a visual memory architecture for manipulation, and MemoryVLA combines working-memory tokens with a bank that preserves visual detail and higher-level semantics (Shi et al., 2026). MAP-VLA (Li et al., 2025) retrieves task-stage prompts learned from demonstrations. Notes-to-Self (Haresh et al., 2026) externalises object locations, plans, and subgoal progress in a language scratchpad. These systems make memory content more explicit, but they introduce a bank, retrieval rule, or language-generation channel in addition to the recurrent policy state.

MemBodied occupies a different position in this design space. It neither retains a growing sequence of observations nor retrieves discrete memory entries. Instead, it represents episode history through a set of layer-wise associative matrices and an initial-scene anchor, both with storage independent of episode length, and conditions the action expert on their readouts.

### A.3 Fast-weight associative memory

Fast-weight models separate slowly optimised parameters from rapidly updated episode state. Schmidhuber (1992) introduced learned control of fast-weight memories. Linear attention was later interpreted as an associative fast-weight programmer (Schlag et al., 2021). DeltaNet replaced purely additive key-value writes with a delta rule that corrects the association already recalled for a key, and Gated DeltaNet added data-dependent retention (Yang et al., 2025). Most directly related, $\delta$-mem augments a language model with a fixed-size online matrix whose readout produces low-rank attention corrections (Lei et al., 2026).

MemBodied transfers the associative-state principle to continuous robot control. Rather than perturbing attention queries or outputs, it injects a projected memory vector into a dedicated suffix token before attention. We use this interface comparison to analyse how an associative state is most effectively exposed to the action network.

### A.4 Memory-focused evaluation

Memory-specific manipulation benchmarks have recently proliferated. MemoryBench evaluates spatial memory and action recall, requiring policies to recover scene information or actions from earlier in an episode (Fang et al., 2025). MIKASA-Robo broadens this scope through a diverse set of partially observable tabletop tasks designed to test memory-intensive control (Cherepanov et al., 2026a). RMBench comprises nine dual-arm manipulation tasks organised by memory complexity, ranging from retaining information from a single earlier stage to accumulating evidence across multiple observations or attempts (Chen et al., 2026). We use RMBench as our primary memory evaluation benchmark because its controlled $M(1)$ and $M(n)$ categories distinguish tasks that require retaining a single earlier event from those that require accumulating information across several events within a common bimanual manipulation setting. In contrast, LIBERO is not designed specifically around episodic recall; its spatial, object, goal, and long-horizon suites provide a complementary test of whether a memory-augmented policy remains effective on general manipulation (Liu et al., 2023).

## B Additional Experimental Details

This section describes the evaluated tasks, training configurations, baseline implementations, and inference-efficiency profiling. Table 3 summarises the shared training configurations and hyperparameters.

Table 3: **Training configuration and Hyperparameters.**

| Setting | Configuration |
|---|---|
| Visual input | Three camera streams, resized with padding to $224 \times 224$ |
| Action generation | 50-step action horizon; 10 flow-matching denoising steps |
| Numerical settings | bfloat16; seed 42; EMA 0.99; global gradient clipping at 1 |
| Optimiser | AdamW, $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, weight decay $10^{-10}$ |
| Learning-rate schedule | Cosine decay with 1,000 warm-up steps, peak rate $2.5 \times 10^{-5}$, and terminal rate $2.5 \times 10^{-6}$ |

### B.1 Evaluated RMBench tasks

Our evaluation uses five RMBench tasks that cover both memory-complexity categories (Chen et al., 2026): three $M(1)$ tasks and two $M(n)$ tasks.

$M(1)$ **tasks.** These tasks require information from a single prior event in the observations. In **Put Back Block**, the robot moves a block from one of four pads to the centre, presses a button, and returns it to its original pad, requiring it to remember which pad was initially occupied. In **Rearrange Blocks**, the robot moves the block between two pads onto the empty pad, presses a button, and then moves the block that originally occupied a pad into the middle, requiring it to retain

the initial layout. In **Swap Blocks**, the robot exchanges two blocks placed on two of three trays using the empty tray as temporary space, requiring it to remember the initial block-tray assignment and block position throughout the transfers.

**$M(n)$ tasks.** These tasks require accumulating information across multiple prior events which are specifically attempts in this case. In **Battery Try**, the robot inserts two batteries in different orientations until the dashboard indicates the correct combination, requiring it to remember which combinations have already been tested. In **Block Ranking**, the robot rearranges three coloured blocks and presses a button to test each ordering until the correct one is found, requiring it to remember which orderings have already failed.

### B.2 RMBench configuration

The RMBench models adapted the PaliGemma backbone using LoRA rank 16 and alpha 16, and the action expert using rank 32 and alpha 32. Models were trained for 10,000 steps with batch size 8. Gradients propagated through the complete sampled sequence. Consecutive sequence elements were separated by 50 environment steps, matching one action horizon. The sequence length was 8 for Put Back Block, Rearrange Blocks, and Swap Blocks; 14 for Battery Try; and 18 for Block Ranking, determined based on average episode lengths in the task. he main MemBodied configuration used associative rank $r = 128$ and $\alpha_{\text{mem}} = 256$, giving an explicit retrieval multiplier of $\alpha_{\text{mem}}/r = 2$.

### B.3 LIBERO Configuration

We trained the LIBERO experiment differently from the RMBench models using full fine-tuning to match the published $\pi_0$ baseline setup. Training used a batch size of 32, 30,000 steps, and a sequence length of six. Evaluation used five-step replanning, following the official $\pi_0$ LIBERO setup. To match this replanning cadence while retaining the 50-step memory-update interval used during training, we maintained ten recurrent memory slots in a round-robin schedule. Each five-step policy call reads one slot, and that slot was revisited after 50 environment steps. At the revisit, the delayed write used the new observation together with a summary of the complete 50-step action chunk predicted on the slot's previous visit. This summary therefore includes predicted actions beyond the five steps executed before the next replan.

### B.4 Baseline implementations

Table 4 records how each evaluated baseline was implemented. Diffusion Policy, ACT, X-VLA, and the $\pi_{0.5}$ baseline in Table 1 are published RMBench results (Chen et al., 2026).

Table 4: **Implementation details for locally evaluated baselines.** "Local adaptation" indicates that the method was implemented and trained within our $\pi_0$ evaluation setup.

| Method | Type | Implementation |
|---|---|---|
| $\pi_0$-Stateless | Local baseline | Current observation only; batch size 32; 10,000 steps. |
| $\pi_0$-FrameStack | Local baseline | Four observations separated by 50 environment steps and concatenated in the policy context; samples with insufficient history were padded to four frames. |
| $\pi_0$-$\mu$-VLA | Local adaptation | 64 recurrent memory tokens; sequence length 8; stride 1; gradients truncated every two recurrent steps. |
| $\pi_0$-Vanilla Recurrent Memory | Local adaptation | Rank 128 and scale 256; query/output attention corrections; recurrent hidden-state writes. |
| NativeMEM | Local adaptation | Memory-tokenizer training for 50,000 steps, offline token caching, then policy training for 20,000 steps with queue stride 1. |
| NativeMEM + MemBodied | Combination | NativeMEM pipeline above with MemBodied sequence length 8 and sequence stride 50. |
| $\pi_0$-Hint | Inference-only reference | Simulator-derived textual task-progress hints appended at inference, without retraining. |

The $\pi_0$-Hint annotations exposed only task progress relevant to the evaluated memory demands. For Put Back Block, they identified the block's original mat and whether it had been moved, the button pressed, and the block returned. For Rearrange Blocks, they recorded whether the first block had reached the empty mat, the button had been pressed, and the second block had been moved between the mats. For Swap Blocks, they described completion of the three block-transfer stages and the button press. Battery Try listed previously attempted polarity combinations and whether the dashboard was on, while Block Ranking listed colour orders attempted before earlier button presses. Because the stateless policy was not trained to consume these annotations, $\pi_0$-Hint is a privileged-information reference rather than an upper bound.

The internal variants changed how the associative state affected action generation. MEMBODIED-AS projected each retrieval into additive corrections to the attention query and output representations. MEMBODIED-H augmented the gated-delta state with a slower LSTM-like recurrent cell controlled by a learned forget gate. The latter therefore changed the state dynamics as well as its use by the action network, so these results compare complete memory designs rather than isolated readout substitutions.

Unlike the other locally trained RMBench baselines, the NativeMEM baseline followed a separate staged training procedure. A memory tokenizer was first trained on the RMBench mixture for 50,000 steps, after which its per-frame, per-view tokens were cached offline for each target task. The tokenizer was then frozen while the queue-conditioned policy was trained for 20,000 steps. The standalone model used a queue stride of one; the combined NativeMEM + MEMBODIED model retained this queue while training the associative state with sequence length 8 and stride 50.

### B.5 MEMORY VARIANTS

The reported attention-steering and hierarchical variants use the same associative memory configuration as MEMBODIED.

**Attention steering (MEMBODIED-AS).** At each action-expert layer, the state token queries the associative matrix. Separate learned projections turn the readout into additive corrections to the action tokens' attention queries and projected attention outputs. The readout is scaled by $\alpha_{\mathrm{mem}}/r$, and each correction is scaled by the root-mean-square magnitude of the representation it modifies.

**Hierarchical memory (MEMBODIED-H).** This variant retains the attention-steering readout but adds a second, slower matrix $C_t^{(l)}$ to each layer's recurrent state. The gated-delta write produces an updated associative matrix for each value slot. A sigmoid forget gate derived from the post-layer state token blends their average with the previous slow matrix $C_{t-1}^{(l)}$ to obtain $C_t^{(l)}$. The final associative state is a weighted average of each gated-delta matrix and $C_t^{(l)}$. Thus, MEMBODIED-H changes the state update as well as the readout relative to the main memory-token model; the variants are complete designs, not isolated readout substitutions.

### B.6 $\pi_{0.5}$ ADAPTATION

Unlike $\pi_0$, $\pi_{0.5}$ represents robot state in the language prefix and has no dedicated state token in the action-expert suffix. We therefore use the memory token's hidden representation for both associative-memory operations. At each action-expert layer, the token's pre-attention representation is projected to form the read query, and the gated retrieval is then added to that token before attention. Its post-layer representation is projected to form the write key for the subsequent memory update, whose value combines the action summary with the following visual observation. The token is recreated on each call, whereas the associative matrices persist across calls within an episode. The initial-scene anchor is retained by fusing its readout into the action tokens.

### B.7 INFERENCE-EFFICIENCY PROFILING

We profiled MEMBODIED and NativeMEM in bfloat16 on NVIDIA A100 GPUs with 80 GB of VRAM, using one GPU per evaluation run. We measured model-side computation only. For MEMBODIED, timing covered policy calls from invocation until the action array was returned. This

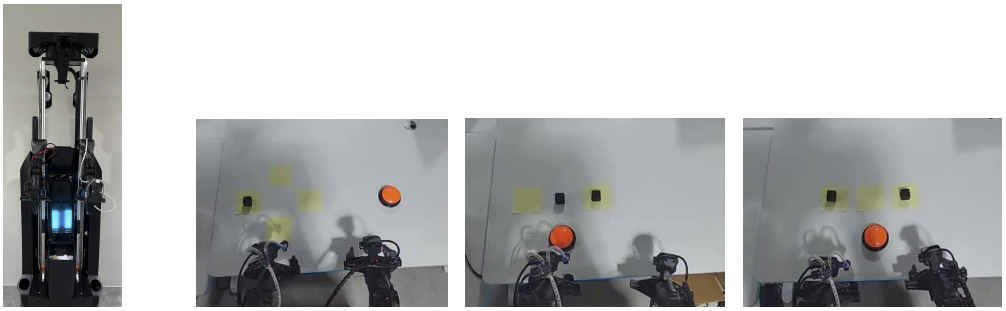
includes policy-side input preprocessing, current-observation vision encoding, associative-memory reads and updates, and generation of a 50-step action chunk using 10 flow-matching denoising steps. For NativeMEM, per-cycle latency was the sum of the model-side time for all video history encoding calls since the preceding policy query and the subsequent policy inference call. This includes per-frame video-history encoding, updates to the expanding history queue and policy context, and action generation.

We evaluated Put Back Block, Rearrange Blocks, Swap Blocks, and Battery Try for 50 episodes per task and policy, yielding 200 episodes per policy. The first three policy queries of each task-policy run were treated as warm-up and excluded. The reported steady-state latency is the arithmetic mean over all retained policy cycles pooled across the four tasks, comprising 2,548 NativeMEM cycles and 2,563 MEMBODIED cycles. Peak memory is the maximum JAX allocator `peak_bytes_in_use` observed across all calls and tasks. NativeMEM required 1,593.7 ms per policy cycle and 20.57 GiB of peak GPU memory, whereas MEMBODIED required 129.2 ms and 18.61 GiB, corresponding to 91.9% lower latency and 9.5% lower peak memory.

# C REAL-WORLD DATA AND ROBOT EVALUATION

## C.1 ROBOT DETAILS

The robot comprised two AgileX PiPER arms, an Orbbec Gemini 336L top camera, and two Intel RealSense D405 wrist cameras (Figure 5). The robot state and action vectors each contained 14 values: six joint values and one gripper value for each arm.



**(a)** Robot **(b)** Put Back Block. **(c)** Rearrange Blocks. **(d)** Swap Blocks.

**Figure 5: Real-robot platform and representative task configurations.** The task panels show top-camera observations approximately one second after the start of a cleaned demonstration. They illustrate the initial workspace configurations and are not evaluation rollouts.

## C.2 DATASET COLLECTION AND PREPROCESSING

We manually collected 50 demonstrations for each of the three tasks. Each demonstration contained synchronised top, left-wrist, and right-wrist views recorded at 30 Hz, together with robot states and actions. Table 5 summarises the data retained after cleaning. The cleaned top-camera videos have resolution $330 \times 240$, while both wrist-camera streams have resolution $424 \times 240$.

Table 5: **Real-world demonstration data after cleaning.** Mean duration is computed from the retained 30 Hz trajectories.

| Task | Collected | Retained | Frames | Mean duration (s) |
|---|---|---|---|---|
| Put Back Block | 50 | 41 | 54,228 | 44.1 |
| Rearrange Blocks | 50 | 50 | 48,914 | 32.6 |
| Swap Blocks | 50 | 49 | 86,818 | 59.1 |

Demonstrations affected by recording faults were removed before training. The retained trajectories were temporally trimmed to the demonstrated interaction, and the recorded joint states were

denoised. The top-camera stream was cropped to the task workspace during preprocessing. Task-specific real-robot policies were trained on the corresponding cleaned datasets using a training recipe similar to the RMBench experiments.

### C.3 Evaluation protocol

We evaluated each policy for 20 physical rollouts on each task. A rollout was counted as successful only when the complete instructed task was achieved, and partial completion received no credit. The recurrent associative state and initial-scene anchor were reset at the start of every rollout. The resulting success rates are reported in Figure 3b.

## D Additional Analysis

### D.1 Additional baseline Clarification

Table 1 shows that our $\pi_0$-$\mu$VLA adaptation achieved 0.0% success on all five RMBench tasks under the shared training budget. This result shows only that the adaptation did not learn an effective recurrent policy under our training recipe; it should not be interpreted as a general comparison with $\mu$VLA under its original setting. One possible explanation for the poor performance in our adaptation is that learning recurrence through carried tokens may require longer training. The inference-time $\pi_0$-Hint policy reached 2.0% mean success. Because the policy was not trained to consume its simulator-derived textual hints, this condition is a privileged-information reference rather than a trained oracle or an upper bound.

### D.2 Memory Rank

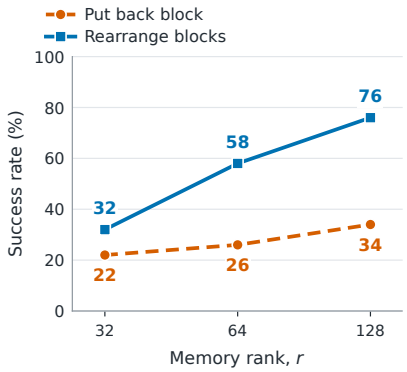



**Figure 6:** Associative memory rank comparison in the anchor-free memory-token variant.

In the anchor-free memory-token variant, success increases monotonically as the rank grows from 32 to 64 to 128 on both Put Back Block and Rearrange Blocks (Figure 6). Their two-task mean rises from 27.0% to 42.0% and 55.0%, respectively. The policy's sensitivity to rank is consistent with associative-state capacity influencing action selection: a larger state can preserve a richer set of episode-dependent associations for later retrieval.

### D.3 Initial-Frame Anchor Representation

We also tested a simpler way to provide the initial scene: adding the episode's first frame to the policy prefix alongside the current input, rather than using the fixed episode-anchor pathway. On Put Back Block task, this first-frame-prefix variant achieved 22.0% success, compared with 34.0% for the memory-token model without an anchor and 40.0% with the proposed anchor. This could be because adding the full first frame introduces distracting visual tokens into the policy context, whereas the anchor selectively retrieves a compact reference to the initial scene.

### D.4 Cross-Episode Memory Carryover

Table 6: RMBench success rates (%) with associative memory reset or carried across episodes. The initial-frame anchor is reset at the start of every episode in both conditions.

| Task | Memory reset | Memory carried over |
|---|---|---|
| Put Back Block | 40.0 | 18.0 |
| Rearrange Blocks | 92.0 | 78.0 |
| Swap Blocks | 56.0 | 36.0 |
| Battery Try | 40.0 | 24.0 |
| Block Ranking | 22.0 | 28.0 |
| Mean | 50.0 | 36.8 |

As an exploratory test, we carry the associative state across rollout boundaries, although MemBodied was trained with the state reset at the start of each episode. The initial-frame anchor is still reset for each new rollout. Table 6 shows that this reduces mean success from 50.0% to 36.8%, with lower success on four of the five tasks. The largest declines occur on Put Back Block and Swap Blocks (22 and 20 percentage points), where the relevant location or progress must be established within the current episode. Block Ranking is the exception, improving by 6 points. Its trial-and-error structure may respond differently to carried state, although this increase does not establish that information from earlier episodes was useful. Overall, the results do not support a general benefit from carrying over memory under this evaluation, as the declines are consistent with information from earlier episodes interfering with episode-specific decisions.

## E Qualitative Analysis

The mean success rates show that MemBodied improves memory-dependent control, but do not identify whether the gains arise at the decisions that require history. We therefore manually annotate all 50 paired rollouts of $\pi_0$-Stateless and MemBodied, using identical seeds, and identify the stages at which their behaviours diverge. The analysis separates RMBench tasks that require recalling one earlier observation, denoted $M(1)$, from tasks that require accumulating the outcomes of several attempts, denoted $M(n)$. Across both groups, the policies behave similarly when the current observation is sufficient, but diverge once the correct action depends on information that is no longer visible.

### E.1 Recalling an earlier observation

#### E.1.1 Put Back Block

Put Back Block isolates spatial recall: both policies move the block to the centre and press the button in every rollout, but they differ when deciding where to return it. The stateless policy returns to the wrong pad in 37/50 rollouts. Its 13 correct choices are close to the 13.8 expected from its own pad preference, indicating that its return location does not track the episode's initial pad. MemBodied reduces wrong-pad returns to 13/50 and corrects 24 stateless errors without introducing the reverse error on any paired seed. Its 37 correct choices also exceed the 14.5 expected by chance. Both comparisons are significant at $p<10^{-4}$. The benefit of MemBodied is therefore concentrated on the history-dependent choice of return location, rather than the preceding pick-and-place sequence.

Aside from the recall errors, the remaining failures expose two execution constraints. Of the 17 correct returns towards the front pad, 14 reach the pad but stall instead of releasing the block. The stateless policy exhibits the same tendency in 8/9 relevant rollouts, suggesting a shared placement difficulty rather than a memory-specific failure. In addition, six of MemBodied's eight failures classified as residual recall errors occur when the block starts on the right: the policy repeats its dominant left-arm motion towards the front pad instead of switching arms. These two patterns account for 20 of the 30 MemBodied failures. Thus, memory largely resolves the target location recall problem, but not the motor strategy needed to reach every target reliably.

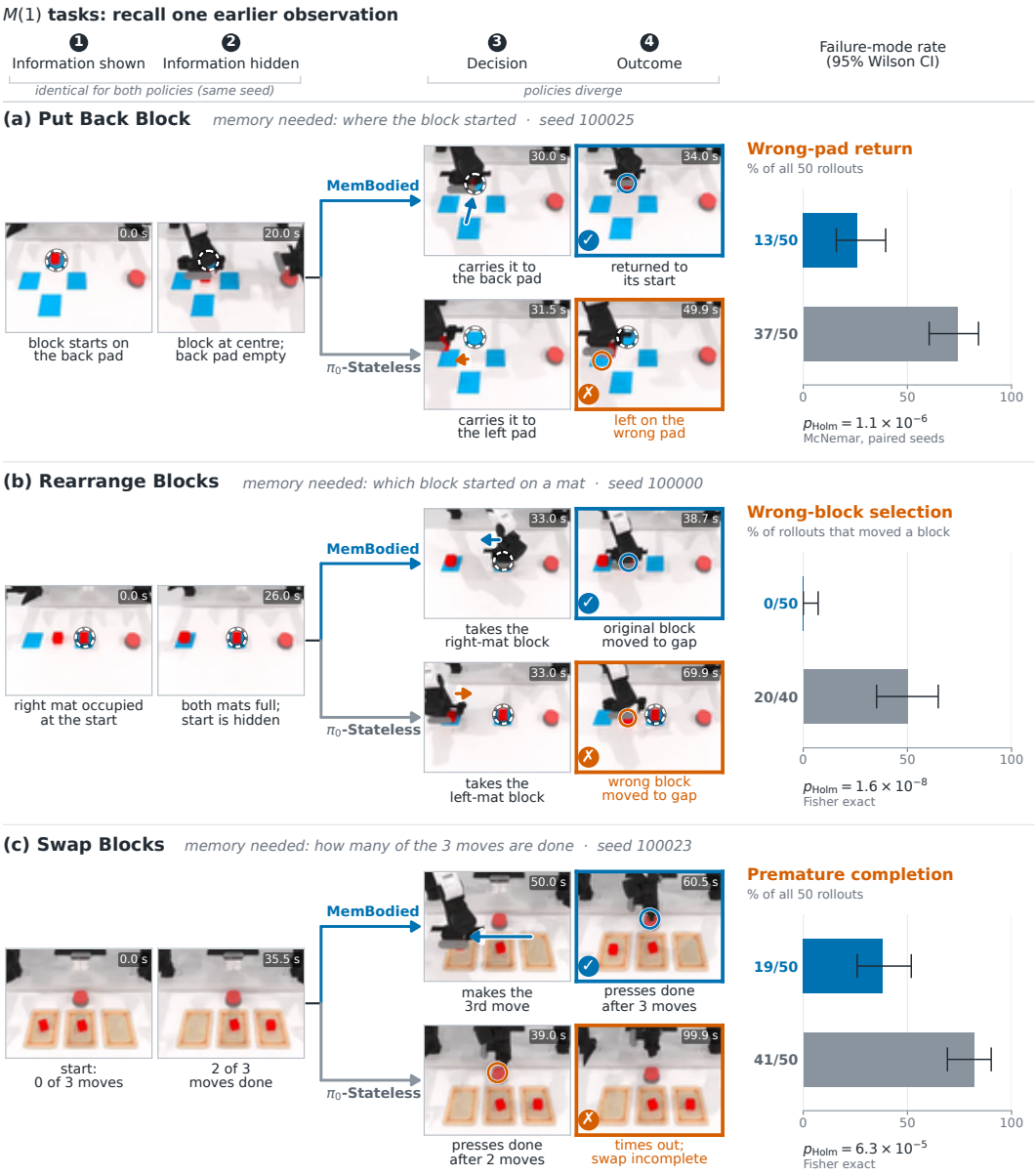



**Figure 7: Failure modes on the three $M(1)$ tasks.** Representative paired rollouts show the same seed as task-relevant information becomes hidden and the policies reach the ambiguous decision. The blue trajectory corresponds to MEMBODIED and the grey trajectory to $\pi_0$-Stateless. Bars summarise the annotated failure mode across rollouts, with marginal 95% Wilson confidence intervals. Put Back Block and Swap Blocks use all 50 rollouts per policy; Rearrange Blocks is conditioned on rollouts in which a block is moved (50 for MEMBODIED, 40 for $\pi_0$-Stateless). The $p$-values shown in the figure are Holm-corrected.

### E.1.2 REARRANGE BLOCKS

Rearrange Blocks tests recall through object selection rather than target selection. After the first placement and button press, both mats contain identical blocks, so only the initial layout identifies which block should be returned to the gap. Both policies complete the shared early stages in every rollout, but their subsequent choices differ. MEMBODIED selects the correct block specified by the initial layout in all 50 rollouts. The stateless policy instead exhibits a strong left-block preference: it chooses the left block in 37/40 rollouts in which it moves a block, selects the correct block only 20 times, and moves no block in the remaining ten rollouts. This shows that memory changes the episode-specific decision made after the two scenes become visually indistinguishable, rather than merely improving the manipulation stages.

Object selection does not explain the entire performance gap. Of the 20 stateless rollouts that select the correct block and bring it to the target pad, only one succeeds, whereas MEMBODIED succeeds in 46/50 rollouts overall. The stateless policy also does not press the button fully, leading to task failures. These execution errors occur after the history-dependent object selection and therefore compound, rather than explain, the recall gap.

#### E.1.3 SWAP BLOCKS

Swap Blocks differs from the previous tasks because the hidden variable is progress rather than an initial location. Exchanging two identical blocks requires three moves through an empty pad, after which the visible occupancy matches the start of the episode. Intermediate occupancy patterns can also be valid starting layouts. Consequently, the current image does not reveal whether the policy should move another block or press the completion button.

Both policies can perform the required manipulation: MEMBODIED makes at least one move in 48/50 rollouts and the stateless policy in 41/50. The main difference is in deciding when the swap is complete. The stateless policy presses the completion button prematurely in 41/50 rollouts, compared with 19/50 for MEMBODIED, and completes all three moves in only eight rollouts, compared to 31 for MEMBODIED. Among rollouts in which the button is pressed, MEMBODIED waits until the swap is complete in 28/47 cases, whereas the stateless policy does so in only 6/47. All three progress-related comparisons remain significant ($p<10^{-4}$). This pattern is consistent with memory tracking progress through the move sequence. However, the tracking is not perfect: 12 of the 22 residual MEMBODIED failures press after two moves, making the final transition the dominant remaining error.

### E.2 ACCUMULATING OUTCOMES ACROSS ATTEMPTS

#### E.2.1 BATTERY TRY

Battery Try requires the policy to change its next action after transient failure feedback. The two policies set both batteries at similar rates (23/50 for MEMBODIED and 20/50 for $\pi_0$-Stateless), indicating comparable grasp-and-insert behaviour before memory is needed. They diverge after an unsuccessful orientation test, once the meter has returned to rest and the current image no longer reveals what was tried. The stateless policy makes a median of four further attempts and completes the task in 8/20 rollouts; MEMBODIED usually succeeds within one further attempt and resolves it in 20/23. This difference remains significant after Holm correction ($p=0.028$), localising the benefit of MEMBODIED to the point where an earlier outcome must alter the next action. The dominant remaining limitation is grasp stability: 25 of the 30 MEMBODIED failures involve dropping a battery during pickup or insertion, a common failure mode in both policies.

#### E.2.2 BLOCK RANKING TRY

Block Ranking Try extends the same dependence on earlier outcomes to a longer search. Both policies complete an initial block arrangement at similar rates (38/50 for MEMBODIED and 40/50 for $\pi_0$-Stateless), but differ in whether they verify it. MEMBODIED presses the test button after 35/38 completed arrangements, compared with 14/40 for the stateless policy. On four paired seeds, the stateless policy even constructs the same arrangement that MEMBODIED successfully verifies but dismantles it without testing it. Because a completed arrangement looks identical before and after testing, this difference shows that MEMBODIED helps in remembering whether a candidate arrangement still needs to be verified, rather than in the preceding pick-and-place operation.

The remaining errors separate immediate verification from longer-term search. Six of the 14 stateless button presses miss the button, compared with none of the 35 MEMBODIED presses, so motor precision further compounds the stateless failure rate. After a rejected arrangement, however, MEMBODIED builds and tests a new order in only 10/28 rollouts; in other cases, it fails in rearranging the blocks or retests the rejected arrangement. These longer-search errors do not differ from the stateless policy. The differences in immediate verification and button execution both remain significant after Holm correction ($p\leq 0.001$), but the broader pattern shows that memory supports the immediate transition from rearranging to verification more clearly than the full multi-attempt search.

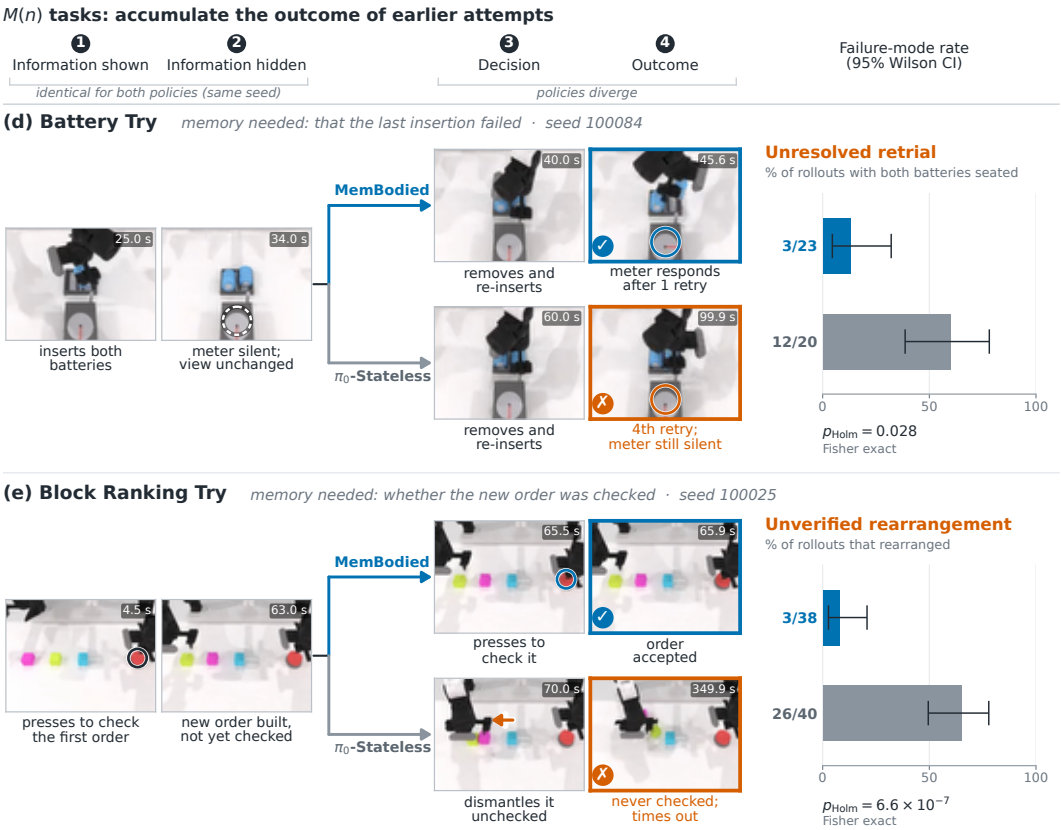



**Figure 8: Failure modes on the two $M(n)$ tasks.** Representative paired rollouts show decisions that depend on the outcome of an earlier attempt. The blue trajectory corresponds to MEMBODIED and the grey trajectory to $\pi_0$-Stateless. Bars report the conditional failure rate with marginal 95% Wilson confidence intervals: unresolved retries among rollouts that seat both batteries (23 for MEMBODIED, 20 for $\pi_0$-Stateless), and unverified arrangements among rollouts that complete a rearrangement (38 and 40, respectively). The $p$-values shown in the figure are Holm-corrected.

Across all five tasks, the rollout analysis localises the performance gains shown in Section 5 to the decisions that require past information in an episode. MEMBODIED does not consistently change the early motor stages shared by both policies. Instead, it reduces the failure specific to each aliased decision: returning to the wrong pad, selecting the wrong block, declaring a swap complete too early, repeating an unsuccessful battery orientation, or dismantling an arrangement before testing it. The remaining failures separate into execution errors, such as stalling, missed presses, and dropped objects, and longer-term memory errors that arise when several updates must be retained. These behavioural patterns are consistent with the recurrent state retaining task-relevant episode information while preserving general robot control ability.