DRAM: DELTA-RULE RECURRENT ASSOCIATIVE MEMORY FOR ROBOT MANIPULATION POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Robotic manipulation is inherently history-dependent, yet most pretrained robotic policies condition on only the current observation or a short temporal window. Equipping such policies with long-term memory remains challenging: existing approaches either feed the backbone multi-frame observation windows, which substantially increase inference cost, or rely on pre-defined semantic features, which limit task generality and may also require the retraining of the backbone to adapt to the memory. We introduce DRAM (Delta-rule Recurrent Associative Memory), a plug-and-play memory module that can be attached to a wide range of pretrained robotic policies, endowing them with long-horizon memory without architectural modification or backbone retraining, requiring only task-specific post-training of the memory module and action expert. DRAM maintains a fixed-size associative memory using gated delta-rule linear attention, with a modified update that incorporates all tokens within each frame in parallel. An architecture-agnostic readout integrates historical context into action prediction across different policy architectures. Experiments show that DRAM consistently improves frozen pretrained policies over short-context baselines and alternative compact memory designs, validating its effectiveness as a fixed-size, post-hoc memory module trained with the backbone frozen.

1 INTRODUCTION

Historical information is a fundamental requirement for robot manipulation over extended horizons (Shi et al., 2026; Li et al., 2026; Jiang et al., 2026). However, many manipulation tasks are only partially observable: the current observation may not reveal objects that were previously visible but have since become occluded, task stages that have already been completed, or the outcomes of earlier interactions. Consequently, visually similar observations may require different actions depending on the preceding history. Therefore, a policy must maintain and retrieve task-relevant history information across time to form a more complete estimate of the environment state and task progress. However, many existing robot policies condition their actions only on the current observation or a short temporal window, preventing historical information outside that window from influencing subsequent decisions and limiting their performance on history-dependent tasks (Li et al., 2026; Jiang et al., 2026).

Recent studies have incorporated history as memory into robot policies. Some retain compact recurrent summaries of past observations: RoboTTT compresses visuomotor history into online-updated fast weights (Jiang et al., 2026), and ReMem-VLA propagates temporal context through frame-level and chunk-level recurrent queries (Li et al., 2026). Others store historical features in an explicit memory bank and retrieve task-relevant entries, as in MemoryVLA (Shi et al., 2026). These approaches differ substantially in how history is represented, updated, and injected into the policy, and are instantiated as components of their respective policy architectures. This motivates a memory mechanism that maintains fine-grained historical information with bounded state size and exposes the retrieved information through an architecture-agnostic interface.

Linear attention (Choromanski et al., 2021) provides a natural foundation for this purpose; it replaces the softmax similarity with a factorizable kernel and uses the associativity of matrix multiplication to aggregate information before applying the queries (Katharopoulos et al., 2020). Under causal processing, this computation admits a recurrent implementation in which historical associations are maintained in a fixed-size matrix (Katharopoulos et al., 2020; Schlag et al., 2021). Consequently, the size of compressed historical associations does not grow with the number of preceding observations.

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.

However, the standard additive update continually superposes new associations onto the existing state, making it difficult to remove previous associations and can eventually cause interference between similar keys (Schlag et al., 2021; Yang et al., 2024).

In this work, we introduce **DRAM**, a **D**elta-rule **R**ecurrent **A**ssociative **M**emory for robot manipulation policies. Building on delta-rule memory management (Schlag et al., 2021; Yang et al., 2024), instead of unconditionally adding each new association, DRAM first retrieves the value currently associated with a key and then writes the residual between this prediction and the new value. Therefore, the memory of DRAM is updated by correcting its existing association rather than repeatedly accumulating the complete value. This formulation enables DRAM to continuously incorporate new observations into a fixed-size recurrent state while controlling redundant and conflicting writes.

Moreover, DRAM operates on visual patch tokens while maintaining recurrent memory updates across frames. At each control step, all visual patches query the shared pre-update memory state carried over from preceding frames. Although they access the same memory snapshot, their distinct queries produce patch-specific historical features, which are fused with the current representations through a zero-initialized residual interface for action prediction. Only after this read operation, the information of the current frame is written into memory using the delta rule, producing the state for the next frame. This *read-before-write* design prevents the current observation from entering its own retrieval and avoids imposing an artificial sequential order among patches within the same image. Besides, implemented as a residual branch, DRAM preserves the original policy backbone and action-generation components, and can be integrated into different robot-policy backbones without architecture-specific redesign of their action decoders. We evaluate DRAM on diverse robot manipulation tasks across multiple policy backbones, including real-robot experiments. The results show that DRAM consistently improves the performance of backbone policies.

Our contributions are summarized as follows:

- We introduce DRAM, a delta-rule recurrent associative memory for robot policies. DRAM maintains historical associations in a fixed-size state and updates them by correcting existing memory predictions rather than unconditionally accumulating new values.
- We develop a patch-level, read-before-write memory mechanism that retrieves information from preceding frames before updating the recurrent state with the current observation, enabling fine-grained historical information to directly influence action generation.
- We provide a lightweight residual interface that makes DRAM plug-and-play across different robot-policy backbones while preserving their original components, and demonstrate consistent improvements across diverse memory-dependent manipulation tasks, including real-robot evaluations.

2 PRELIMINARIES

2.1 PROBLEM FORMULATION

At environment step t, a robot receives a visual observation o_t, a proprioceptive state s_t, and a task instruction ℓ, and predicts an action chunk $\mathbf{a}_t = a_{t:t+C-1}$ of length C. Let X_t denote the current-frame features produced by the policy backbone. A policy without persistent memory predicts

$$\mathbf{a}_t \sim \pi_\theta(\cdot \mid X_t, s_t, \ell). \tag{1}$$

The current inputs may not reveal task-relevant information from earlier interactions. Memory augments this conditioning with a persistent representation of the observation history.

2.2 RECURRENT MEMORY

Recurrent memory carries a state M_t across successive inputs:

$$M_t = F_\psi(M_{t-1}, X_t), \tag{2}$$

where F_ψ integrates the current features with the previous state. With a fixed-size state, the storage required for persistent memory is independent of the number of preceding steps.

Linear attention provides an associative implementation of this recurrence (Katharopoulos et al., 2020; Schlag et al., 2021). For a generic token sequence, let $q_i, k_i \in \mathbb{R}^{1\times d_k}$ and $v_i \in \mathbb{R}^{1\times d_v}$, with feature maps absorbed into the queries and keys. Its unnormalized form admits

$$S_i = S_{i-1} + k_i^\top v_i, \qquad r_i = q_i S_i, \tag{3}$$

where $S_i \in \mathbb{R}^{d_k \times d_v}$ compresses historical key–value associations. This matrix-valued recurrence underlies the memory formulation adopted in this work. Other implementations include gated vector states in LSTMs and memory tokens propagated through Transformer layers (Graves, 2012; Bulatov et al., 2022). Despite different state representations and update operators, these mechanisms share the recursive form in equation 2.

2.3 Delta-Rule Associative Memory

The additive update in equation 3 superposes new associations without explicitly correcting existing ones. DeltaNet instead updates memory using the error of its current value prediction (Schlag et al., 2021; Yang et al., 2024). For the local associative objective $\mathcal{L}_i(S) = \frac{1}{2}\|k_i S - v_i\|_2^2$, one gradient step writes the residual $v_i - k_i S$. A gated formulation additionally applies data-dependent retention (Yang et al., 2025):

$$\begin{aligned} \bar{S}_{i-1} &= \gamma_i S_{i-1}, \\ S_i &= \bar{S}_{i-1} + \beta_i k_i^\top \left(v_i - k_i \bar{S}_{i-1}\right), \end{aligned} \tag{4}$$

where $\gamma_i \in (0, 1)$ controls retention and $\beta_i \in [0, 1]$ controls the correction strength, with unit-normalized keys. Unlike additive accumulation, this update modifies memory according to the discrepancy between stored and incoming associations. The standard recurrence processes tokens sequentially; our method adapts this associative principle to joint updates over the tokens of each environment frame.

3 Method

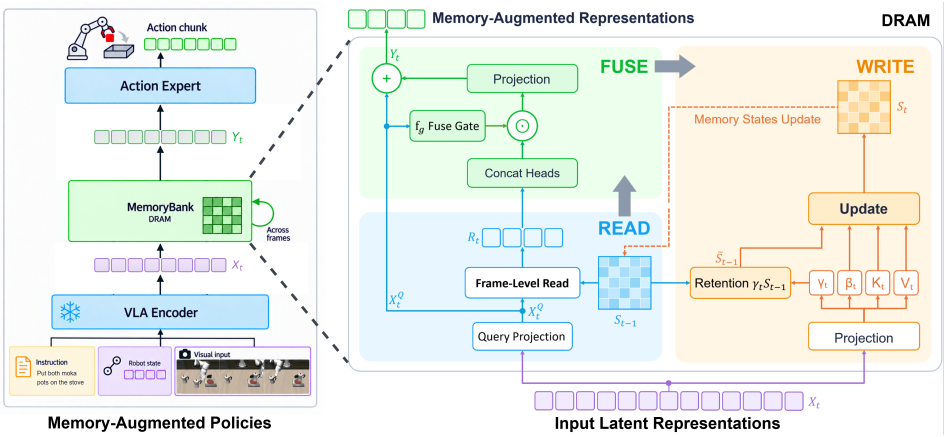

Figure 1: Overview of DRAM. **Left:** DRAM augments representations X_t from a frozen VLA encoder with historical information, producing Y_t for action prediction. **Right:** The memory module follows a read–fuse–write procedure. After reading, current-frame associations update the retained memory through a frame-level delta update, yielding S_t for the next frame.

We introduce the Delta-rule Recurrent Associative Memory (DRAM) module, a frame-level recurrent memory for embodied policies that maintains a fixed-size historical state while preserving the spatial structure of the robot's visual representation and other observations. Our key principle is to decouple the *temporal granularity of memory recurrence* from the *spatial granularity of memory access*: each environment frame constitutes one recurrent step, whereas individual visual patches independently read from and write to a shared associative memory. At every control step, the policy explicitly processes only the current observation, while historical information is carried through a persistent Delta-rule-based memory state. The module consists of three primitives: Read, Write, and Fuse.

3.1 Framework Overview

Figure 1 illustrates the framework of DRAM. At control step t, the frozen VLM backbone encodes the current image, together with other inputs such as the language instruction and the robot state, and exposes a collection of frame-level latent token sequences, denoted collectively by $X_t = \{X_t^Q, X_t^K, X_t^V\}$, whose shapes and granularity are not fixed a priori. We retain the frame-level latent sequences rather than pooling each observation into a single write vector—and we define the query and write sources of one memory layer as views of these latents:

$$X_t^Q \in \mathbb{R}^{M \times D_Q}, \qquad X_t^K \in \mathbb{R}^{N \times D_K}, \qquad X_t^V \in \mathbb{R}^{N \times D_V}, \tag{5}$$

where D_Q, D_K, D_V are the backbone feature dimensions. The three sources follow two interface patterns depending on the host policy's architecture: a shared-source interface sets all three to the same sequence ($M{=}N$), while a separate-source interface draws the write sources X_t^K, X_t^V from the backbone latents X_t with a flexible query source. We present the general case with M query tokens and N write tokens, and treat sharing as a special instance.

Given the normalized sources $Z_t^{\{\cdot\}} = \mathrm{LN}(X_t^{\{\cdot\}})$, the head-wise query, key, and value components are as follows:

$$Q_t^{(h)} = \mathrm{Norm}\big(Z_t^Q W_Q^{(h)}\big), \qquad K_t^{(h)} = \mathrm{Norm}\big(Z_t^K W_K^{(h)}\big), \qquad V_t^{(h)} = Z_t^V W_V^{(h)}, \tag{6}$$

where $h \in \{1, \dots, H\}$ indexes H heads, $\mathrm{Norm}(\cdot)$ denotes row-wise ℓ_2 normalization, and $Q_t \in \mathbb{R}^{H\times M\times d_k}$, $K_t \in \mathbb{R}^{H\times N\times d_k}$, $V_t \in \mathbb{R}^{H\times N\times d_v}$. The rows of K_t and V_t remain paired within each head. Crucially, their row index is *not* an environment timestamp: whether encoding spatial position, camera identity, or language order, all tokens of one frame belong to the same physical observation. They therefore form a single recurrent input rather than a sequence of timesteps—the premise of the frame-synchronous write in Section 3.3.

In the DRAM module, the historical information is carried by a set of H associative memory states per layer,

$$\mathcal{S}_t = \big\{\, S_t^{(h)} \in \mathbb{R}^{d_k\times d_v} \,\big|\, h = 1, \dots, H \,\big\}, \tag{7}$$

whose size is independent of the episode length. Head indices and padding masks are omitted below except where heads interact. With these head-wise representations, the core of memory operation carries out three primitives:

$$\textsc{Read}: \quad R_t = \mathrm{Read}(Q_t,\, S_{t-1}), \tag{8}$$

$$\textsc{Fuse}: \quad Y_t = \mathrm{Fuse}(X_t^Q,\, R_t), \tag{9}$$

$$\textsc{Write}: \quad S_t = \mathrm{Write}(K_t,\, V_t,\, S_{t-1};\, \gamma_t\, \beta_t). \tag{10}$$

READ retrieves historical information for all queries in parallel from the *pre-update* state S_{t-1}. FUSE combines the readout R_t with the query-source tokens to produce the memory-augmented representation Y_t consumed by the downstream policy. WRITE then updates the state for the next frame: it applies the soft retention gate γ_t to the inherited state, then incorporates the current key–value associations through a single joint update modulated by β_t.

These equations specify dependencies rather than a mandatory serial schedule: retrieval reads S_{t-1} rather than the newly written S_t; retention affects only the state-update branch; and fusion depends only on the query source and the historical readout. The raw readout R_t remains available independently of fusion, allowing the policy adapter to decide how retrieved information enters the host policy. We detail the readout path—READ and FUSE—in Section 3.2, the state update—WRITE—in Section 3.3.

3.2 Memory Access and Historical Fusion

At each control step, memory serves the policy before it is updated: the current frame first retrieves history, and the retrieved content is fused into the policy's token stream.

Read: frame-level retrieval from the pre-update state. Before any writing, every query token retrieves from the incoming state:

$$R_t^{(h)} = \frac{Q_t^{(h)} S_{t-1}^{(h)}}{\sqrt{d_k}} \in \mathbb{R}^{M\times d_v}, \qquad R_t = \Big\|_{h=1}^{H} R_t^{(h)}, \tag{11}$$

where $\|$ concatenates along the feature dimension. Queries are computed from the current frame while S_{t-1} contains only past frames, so retrieval conditions on the current observation and task without the current frame leaking into its own readout.

Fuse: zero-initialized gated residual. Fusion combines the concatenated readout R_t with the query source through a gated residual branch,

$$Y_t = \mathrm{Fuse}(X_t^Q, R_t) = X_t^Q + f_O\Big(\sigma\big(f_G(X_t^Q)\big) \odot R_t\Big), \tag{12}$$

where $f_G : \mathbb{R}^{D_Q} \to \mathbb{R}^{Hd_v}$ is a row-wise affine gate and $f_O : \mathbb{R}^{Hd_v} \to \mathbb{R}^{D_Q}$ is the output projection. We zero-initialize f_O, making FUSE an exact identity at initialization; the memory readout thus

initially adds no residual change to the query representations. The gate and projection affect only the query stream, and an optional bidirectional intra-frame mixer (attention and MLP) may further process Y_t within that stream.

From readout to action. How the fused representation is consumed depends on whether the query stream shares its source with the write streams (Section 3.1). With separate sources, the readout is a representation of its own: the query slots' outputs enter the policy as additional context alongside an untouched host token stream (our $\pi_{0.5}$ (Black et al., 2025) host). With a shared source, history is instead carried by the token stream itself: FUSE returns the fused sequence Y_t in place of the raw readout, memory layers stack as in an ordinary transformer with a depth independent of the host, and the action head reads Y_t through its original interface with no modification (our DiT host). In both cases the downstream conditioning interface is unchanged. Appendix B details the two patterns and their host instantiations.

3.3 FRAME-SYNCHRONOUS JOINT WRITE

The original delta rule updates memory states one association at a time, a schedule dictated by the sequential nature of text. In embodied policies, however, all tokens of one frame become available simultaneously, and their serialization, whether spatial layout, camera identity, or language order, carries no temporal meaning. We therefore promote the update unit from a single token to a whole frame. Each environment frame triggers exactly one memory transition, in which all of its key–value associations are written jointly.

Gates: frame-level retention and token-level write strength. A scalar retention gate per head and frame softly decays the inherited state,

$$\gamma_t^{(h)} = \exp\big\{ -\exp(a_h)\,\zeta(\bar{g}_t\, u_{\gamma,h} + b_{\gamma,h})\big\} \in (0,1), \qquad \bar{S}_{t-1}^{(h)} = \gamma_t^{(h)} S_{t-1}^{(h)}, \tag{13}$$

where $\bar{g}_t$ is the mask-aware mean of the write-source rows (Section 3.1), $u_{\gamma,h}$ a learned direction, a_h a learned scale, and $\zeta(\cdot)$ the softplus function. A per-token write gate sets the strength of each association from the corresponding write-source row $g_{t,i}$:

$$\beta_{t,i}^{(h)} = \sigma(g_{t,i} u_{\beta,h} + b_{\beta,h}), \quad \hat{\beta}_{t,i}^{(h)} = \min(\beta_{t,i}^{(h)}, 0.99), \quad w_{t,i}^{(h)} = p_i\, \frac{\hat{\beta}_{t,i}^{(h)}}{1-\hat{\beta}_{t,i}^{(h)}}, \tag{14}$$

with $p_i \in \{0,1\}$ the padding mask. The odds reparameterization $w = \hat{\beta}/(1-\hat{\beta})$ is chosen so that, for a single unit-norm key, the joint update below reduces exactly to the gated delta rule with strength $\hat{\beta}$ (Appendix C.2); the cap bounds $w \le 99$ for numerical conditioning (Appendix C.4).

Update: frame-synchronous joint associative correction. The original delta rule is one gradient step on its prediction error (Sec. 2.3). Instead of applying one such gradient step independently to each token in the latent sequence, we consider resolving all N corrections simultaneously.

Suppressing head indices, let $\bar{S} = \gamma_t S_{t-1}$ and $W_t = \mathrm{diag}(w_{t,1}, \ldots, w_{t,N})$. All N associations of the current frame are written in one transition, defined as the minimizer of

$$S_t = \arg\min_{S \in \mathbb{R}^{d_k \times d_v}} \frac{1}{2}\|S - \bar{S}\|_F^2 + \frac{1}{2}\big\|W_t^{1/2}(K_t S - V_t)\big\|_F^2, \tag{15}$$

where $K_t \in \mathbb{R}^{N \times d_k}$ and $V_t \in \mathbb{R}^{N \times d_v}$ stack the keys and values of the N write tokens row-wise, and $W_t^{1/2} = \mathrm{diag}(\sqrt{w_{t,1}}, \ldots, \sqrt{w_{t,N}})$, so that the second term is simply the weighted sum $\frac{1}{2}\sum_i w_{t,i}\|k_{t,i} S - v_{t,i}\|_2^2$. In the above equation, the first term anchors the update to retained history, and the second asks one candidate state to explain *all* current key–value pairs simultaneously. Since the latter constrains only the directions spanned by the current keys, the anchor—selecting the fitting state closest to $\bar{S}$—keeps all unaddressed content intact.

Being quadratic in S, Equation (15) is minimized where its gradient vanishes, which yields the normal equation $(I_{d_k} + K_t^\top W_t K_t)\, S_t = \bar{S} + K_t^\top W_t V_t$. Since $A \triangleq I_{d_k} + K_t^\top W_t K_t$ is symmetric positive definite for any non-negative weights, the minimizer is unique with the closed-form solution:

$$S_t = A^{-1}\big(\bar{S} + K_t^\top W_t V_t\big) = A^{-1}\bar{S} + A^{-1} K_t^\top W_t V_t. \tag{16}$$

All predictions $K_t \bar{S}$ are evaluated against the *single* retained state, and A^{-1} couples the residual corrections according to the key geometry. Detailed derivation can be found in Appendix C.1.

Note that the decay gate and the joint write are not independent forget mechanisms: the historical term in Equation (16) is $\gamma_t A^{-1} S_{t-1}$, so in directions constrained by current keys the retained content is attenuated beyond the scalar decay—the joint fit replaces it—while unconstrained directions undergo only soft decay.

In practice, A^{-1} is never materialized: writing $S_t = \bar{S} + \Delta$, the normal equation becomes

$$A\,\Delta = K_t^\top W_t \big(V_t - K_t \bar{S}\big) \tag{17}$$

which we solve by Cholesky factorization, returning $S_t = \bar{S} + \Delta$. Fully padded frames leave S_{t-1} untouched; valid frames with all-zero write weights return only the decayed $\bar{S}$. Masks, precision, and gradient flow through the solve are detailed in Appendix C.4.

Why not sequential token writes? One could apply the delta rule token-by-token within a frame, iterating Equation (4) N times. This chains N recurrent states indexed by an arbitrary serialization of the current frame: each correction changes the state used to predict later values, so the final state depends on the within-frame order even for identical inputs (a minimal counterexample is in Appendix C.1). Chunked kernels accelerate this recurrence without removing that dependence. We instead treat one frame as one simultaneous observation: Equation (15) is the same local associative objective as the delta rule, optimized jointly rather than greedily.

Properties. Three properties matter for embodied use; all are proved in Appendix C. *(i) Single-token correspondence.* For a single association (k, v) with $\|k\| = 1$ and weight $w = \hat{\beta}/(1-\hat{\beta})$, Equation (16) reduces to the gated delta rule of Equation (4) with the clipped gate $\hat{\beta}$, so the joint-write rule strictly generalizes delta-rule memory management to simultaneous writes. *(ii) Within-frame permutation invariance.* Applying a common permutation to the rows of K_t, V_t and the diagonal entries of W_t leaves S_t unchanged in exact arithmetic: there is no serialization-induced recency bias among tokens of the same frame. *(iii) Bounded evidence accumulation.* n repetitions of one association act with effective strength $nw/(1+nw)$: evidence accumulates monotonically yet saturates below one, so repeated tokens reinforce rather than dilute the write.

3.4 Training with Truncated Backpropagation through Time

DRAM is designed for post-hoc integration on a pretrained policy: we train the memory module, the policy-specific adapter, and the action expert, while keeping the vision tower and the VLM backbone fully frozen. This choice is deliberate rather than merely economical. The memory is trained jointly with the action expert through truncated BPTT (backpropagation through time). Co-training the deep VLM with the memory would chain gradients through the solve, the backbone layers, and the memory states of earlier frames—brittle to optimize and prone to destabilizing the pretrained representations the policy depends on. The frozen backbone ensures the stability of the latent feature space, allowing memory to learn what to store, retrieve, and forget *on top of frozen features*, with all memory parameters optimized solely by the downstream control objective; we introduce no auxiliary reconstruction, retrieval, or memory-specific loss.

Control objective. Any host policy objective can supervise the memory. In our setup, the action expert is trained using flow matching under its data-to-noise convention: for a demonstrated action chunk $\mathbf{a}_t$, $\epsilon \sim \mathcal{N}(0, I)$, and flow time $\tau = 0.999\, b + 0.001$ with $b \sim \mathrm{Beta}(1.5, 1)$,

$$\mathbf{a}_t^\tau = (1-\tau)\,\mathbf{a}_t + \tau\,\epsilon, \qquad u_t = \epsilon - \mathbf{a}_t, \qquad \mathcal{L}_{\mathrm{FM}} = \mathbb{E}\big\| \, v_{\theta,\psi}(\mathbf{a}_t^\tau, \tau;\, c_t, R_t) - u_t \, \big\|_2^2, \tag{18}$$

using the host policy's padding-aware action reduction. Here c_t denotes the current-frame context, θ the action-expert parameters, and ψ the memory and adapter parameters; R_t collectively denotes the memory readouts consumed by the adapter. Gradients of $\mathcal{L}_{\mathrm{FM}}$ flow through every memory operation—patch-level retrieval, gated writing, and temporal retention—so the contents of memory are shaped directly by their utility for control rather than by any hand-designed storage criterion.

Recurrent unrolling. Training unrolls consecutive frames within a TBPTT window. Inside a window, gradients propagate through historical reads, the retention and write gates, fusion where used, and the differentiable joint solve (Appendix C); recurrent states are detached only at window boundaries, not at activation-checkpoint segment boundaries. Valid-frame losses are averaged over the contributing ranks and accumulated windows. At deployment, each new observation triggers exactly one read–write transition, the action expert denoises only when a new action chunk is required,

Table 1: Success rates (%) on LIBERO. Bold indicates the best result in each row, including ties. Superscripts indicate the backbone; DRAM uses $\pi_{0.5}$. Averages are recomputed over the four suites. Published results use their respective training and evaluation protocols.

Environment	MemoryVLA	HAMLET	μVLA	ContextVLA$^{\pi_0}$	ContextVLA$^{\pi_0\text{-FAST}}$	ContextVLA^{G}	DRAM (Ours)
LIBERO-Spatial	98.40	**99.00**	93.00	97.40	98.30	98.40	**99.00**
LIBERO-Object	98.40	**100.00**	99.40	98.20	99.20	99.00	**100.00**
LIBERO-Goal	96.40	**99.20**	96.60	96.40	95.60	97.20	98.00
LIBERO-10	93.40	92.20	95.80	93.80	90.20	93.40	**99.00**
Average	96.65	97.60	96.20	96.45	95.83	97.00	**99.00**

and the memory persists across chunks and resets across episodes. Note that a bounded persistent state does not bound training-time activation memory, which still grows with the TBPTT window length.

4 Experiments

4.1 Setup

Benchmark. Our main evaluation uses the four LIBERO suites (Liu et al., 2023): LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-10. These suites cover variations in spatial arrangements, objects, and task goals, as well as longer manipulation tasks. Note that LIBERO-10 provides a particularly relevant setting for examining performance over longer execution horizons.

Compared methods. We compare DRAM with representative approaches that incorporate historical information into robot policies: MemoryVLA (Shi et al., 2026), HAMLET (Koo et al., 2026), μVLA (Cherepanov et al., 2026), and ContextVLA (Jang et al., 2025). These methods employ explicit memory banks, historical token aggregation, recurrent memory tokens, or compressed multi-frame context. For ContextVLA, we include its reported results with π_0, π_0-FAST, and GR00T N1.5. DRAM uses $\pi_{0.5}$ as its backbone policy in the main comparison.

4.2 Main Results

Table 1 summarizes the results. DRAM achieves an average success rate of 99.00%, the highest among the compared methods, exceeding the next-best average of 97.60% from HAMLET by 1.40 percentage points. The largest advantage appears on LIBERO-10, where DRAM reaches 99.00%, exceeding the next-best result of 95.80% from μVLA by 3.20 percentage points. This result is consistent with the intended role of DRAM in supporting decisions over longer task executions. On LIBERO-Spatial and LIBERO-Object, DRAM achieves 99.00% and 100.00%, respectively, matching the best reported results in the comparison. On LIBERO-Goal, DRAM obtains 98.00%, below HAMLET's 99.20%, indicating that its advantage is not uniform across all suites.

These reported results indicate strong overall LIBERO performance. Because the published methods use different training and evaluation protocols, we next compare each backbone with and without DRAM to assess its contribution under matched conditions.

To examine whether DRAM benefits different policy architectures, we compare each host policy with its DRAM-augmented counterpart on LIBERO. Note that this requires different integration patterns, which are detailed in Appendix B. As shown in Table 2, DRAM improves the DiT policy across all four suites, increasing its average success rate from 84.31% to 94.63%. The largest improvement occurs on LIBERO-10, where success increases from 75.00% to 91.00%, a gain of 16.00 percentage points.

Table 2: LIBERO success rates (%) across policy backbones. Bold indicates the better result within each pair, including ties.

	DiT		$\boldsymbol{\pi_{0.5}}$	
Suite	Baseline	+ DRAM	Baseline	+ DRAM
LIBERO-Spatial	83.25	**92.00**	97.00	**99.00**
LIBERO-Object	88.00	**97.50**	99.00	**100.00**
LIBERO-Goal	91.00	**98.00**	**98.00**	**98.00**
LIBERO-10	75.00	**91.00**	96.00	**99.00**
Average	84.31	**94.63**	97.50	**99.00**

DRAM also benefits the stronger $\pi_{0.5}$ backbone, raising its average success rate from 97.50% to 99.00%. It improves performance on Spatial, Object, and LIBERO-10, while maintaining the 98.00% success rate on Goal. Together, these results support the applicability of DRAM to both evaluated backbones, with gains observed at different levels of baseline performance.

Table 3: Success rates (%) of different DRAM configurations on LIBERO with $\pi_{0.5}$ from LeRobot. Averages are recomputed over the four displayed suite-level success rates. Bold indicates the best result in each row.

	Baseline	DRAM (Ours)		
Suite	$\pi_{0.5}$	**Native KV**	**Hidden Tokens**	**Last Hidden**
LIBERO-Spatial	97.00	**99.00**	98.50	98.50
LIBERO-Object	99.00	**100.00**	99.00	99.50
LIBERO-Goal	98.00	98.00	**99.50**	96.00
LIBERO-10	96.00	**99.00**	97.50	96.00
Average	97.50	**99.00**	98.63	97.50

4.3 Flexibility of Memory Integration

DRAM separates recurrent memory operations from the policy representations used to construct memory. We examine this flexibility on $\pi_{0.5}$ through three configurations that use different write sources while retaining the same memory update rule and prefix-based injection interface.

Integration configurations. *Native KV* uses the pre-RoPE keys and values from each aligned VLM layer as memory write sources. *Hidden Tokens* derives memory keys and values from the hidden tokens of the corresponding VLM layer. *Last Hidden* uses the final VLM hidden sequence as the write source across memory layers and exposes only the final memory readout. In all configurations, the retrieved features are projected into additional prefix keys and values for the action expert.

Results. As shown in Table 3, Native KV achieves the best average success rate of 99.00%, exceeding the baseline by 1.50 percentage points. Hidden Tokens reaches 98.63%, while Last Hidden matches the baseline average of 97.50%. Thus, the choice of write source affects the benefit of DRAM even with the same memory update rule.

4.4 Sensitivity to Memory Reset Interval

To test sensitivity to interruptions of its recurrent state, we evaluate DRAM on LIBERO-10 while resetting all memory states every L observation-frame updates, with $L \in \{8, 32, 128, 256\}$. The full-memory setting resets only between episodes. We use the same trained checkpoint and inference settings, with 30 rollouts per subtask (300 total).

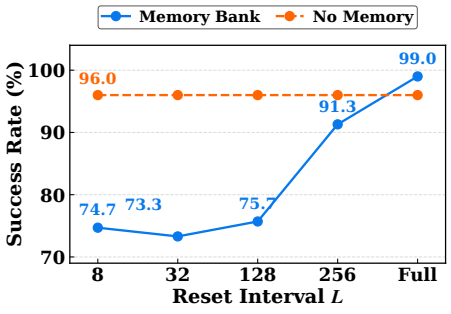

Figure 2: LIBERO-10 success versus memory reset interval L (in frames). Full resets only between episodes; the dashed line denotes the $\pi_{0.5}$ baseline without DRAM.

As shown in Figure 2, resetting memory every 8,32, or 128 frames yields success rates of approximately 73–76%, well below both full memory (99.0%) and the $\pi 0.5$ baseline without DRAM (96.0%). Increasing the interval to 256 frames raises success rate to 91.3%, although it remains 7.7 percentage points below full memory and even still below the no-memory baseline. These results show that the trained policy is sensitive to interruptions of its recurrent state and that longer intervals substantially reduce the performance loss. Periodic resets also alter the memory states encountered later in an episode, so these results characterize the policy's sensitivity to interrupted memory rather than establish a precise effective memory horizon.

4.5 Real-world Experiments

Setup. Using the same DiT-based policy with cross-attention conditioning, we evaluate insertion and a longer, multi-stage task with and without DRAM. Each task-policy setting is evaluated in 20 trials; all other training and evaluation settings are unchanged. Details are provided in Appendix D.

Table 4: Real-world success rates (%).

Task	DiT	DiT + DRAM
Insertion	90.0	**100.0**
Full Task	0.0	**85.0**

Results. As shown in Table 4, DRAM raises insertion success from 90.0% to 100.0% and full-task success from 0.0% to 85.0%. The larger gain on the multi-stage task supports the value of persistent memory in this real-world setting.

5 Related Work

5.1 Memory for Robotic Manipulation

Historical context helps robot policies resolve partial observability and track task progress, and existing approaches differ in how they represent and maintain it. MemoryVLA maintains an explicit bank of perceptual and cognitive features, retrieved via cross-attention with redundant entries consolidated (Shi et al., 2026); HAMLET and ContextVLA instead compress history into compact tokens—moment tokens aggregated over a history window (Koo et al., 2026), or a single context token within the VLM backbone (Jang et al., 2025). Others propagate compact recurrent states: μVLA carries memory tokens through backbone self-attention (Cherepanov et al., 2026), ReMem-VLA maintains frame- and chunk-level recurrent queries (Li et al., 2026), and MEM pairs short-term visual context with a recurrent language summary of longer-term semantics (Torne et al., 2026).

Closest to DRAM, RoboTTT scales visuomotor context through *fast-weight adaptation* (Jiang et al., 2026), inserting test-time-training layers into the action head that encode history by updating small networks during execution. DRAM instead maintains an associative memory of token-level key–value associations, retrieves historical features from the pre-update state, and incorporates each frame through a joint memory update.

5.2 Recurrent Associative Memory

Linear attention admits a recurrent formulation that summarizes historical key–value associations in a fixed-size matrix (Katharopoulos et al., 2020). The connection to fast-weight learning provides an associative interpretation of this state and motivates delta-rule updates that correct existing predictions instead of repeatedly accumulating complete values (Schlag et al., 2021). Subsequent work develops efficient parallel algorithms for DeltaNet and combines associative correction with data-dependent forgetting in Gated DeltaNet (Yang et al., 2024; 2025). These mechanisms provide the foundation for DRAM's memory operations.

Our focus is their use with structured visual observations, where tokens of one frame are simultaneously available, and their serialization carries no temporal meaning. DRAM therefore treats each frame as one recurrent input: queries retrieve from the pre-update state, and current associations are incorporated through a single joint optimization anchored to retained history.

5.3 Frame-wise Delta Updates

Treating one frame as a simultaneous write unit has emerged independently in video modeling. SANA-WM (Zhu et al., 2026) writes each frame into a linear-attention world model explicitly, scaling keys by $1/\sqrt{U}$, while a concurrent analysis (Zhu, 2026) instead solves the per-frame ridge problem, deriving essentially the same closed form and odds-based gate conversion as DRAM. DRAM is concurrent and complementary: the joint write lives in an external, policy-agnostic memory attached to a frozen policy, remains strictly causal at control time, and is trained directly by the control objective under TBPTT.

6 Conclusion

We presented DRAM, a recurrent associative memory module that augments pretrained robot policies with persistent historical context. DRAM maintains a fixed-size memory state with token-level associations, retrieves history from the pre-update state, and incorporates each frame through a joint update that imposes no sequential order among tokens of the same frame. Separating memory operations from policy integration lets DRAM support different representation interfaces while preserving the host policy's components, and experiments on simulated and real-world tasks show consistent gains, most pronounced on longer multi-stage tasks.

References

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy

Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40, 2025.

Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. Recurrent memory transformer. In *Advances in Neural Information Processing Systems*, volume 35, pp. 11079–11091, 2022.

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

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021.

Alex Graves. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pp. 37–45, 2012.

Huiwon Jang, Sihyun Yu, Heeseung Kwon, Hojin Jeon, Younggyo Seo, and Jinwoo Shin. Contextvla: Vision-language-action model with amortized multi-frame context. *arXiv preprint arXiv:2510.04246*, 2025.

Yunfan Jiang, Yevgen Chebotar, Ruijie Zheng, Fengyuan Hu, Yunhao Ge, Jimmy Wu, Tianyuan Dai, Scott Reed, Fei-Fei Li, Yuke Zhu, and Linxi Fan. RoboTTT: Context scaling for robot policies. *arXiv preprint arXiv:2607.15275*, 2026. URL https://arxiv.org/abs/2607.15275.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 2020. URL https://proceedings.mlr.press/v119/katharopoulos20a.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.

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. URL https://arxiv.org/abs/2603.12942.

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.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9355–9366. PMLR, 2021. URL https://proceedings.mlr.press/v139/schlag21a.html.

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 *The Fourteenth International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2508.19236. arXiv:2508.19236.

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.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

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

Haoyi Zhu. Reflections on video DeltaNet. `https://www.haoyizhu.site/blog/video-delta-rule/`, 2026. Blog post (in Chinese). Accessed: 2026-09-23.

Haoyi Zhu, Haozhe Liu, Yuyang Zhao, Tian Ye, Junsong Chen, Jincheng Yu, Tong He, Song Han, and Enze Xie. SANA-WM: Efficient minute-scale world modeling with hybrid linear diffusion transformer. *arXiv preprint arXiv:2605.15178*, 2026.

A The Use of Large Language Models (LLMs)

We employ Large Language Models (LLMs) for grammar checking in our paper.

B Flexible Integration Patterns

DRAM admits the two integration patterns introduced in Section 3.2: separate-source integration, where the query stream is decoupled from the host's tokens, and shared-source integration, where the host's frame tokens themselves carry the memory interface. The choice is dictated by the host architecture rather than by the memory: the READ, WRITE, and FUSE operations of Section 3 are identical in both. Within each pattern, three further choices are orthogonal: where the write sources are tapped, how the readout is delivered to the policy, and how memory layers align with host layers. We instantiate the two patterns with the $\pi_{0.5}$ and DiT hosts used in our experiments.

B.1 Separate-source integration: the $\pi_{0.5}$ host

With designated query slots (learned slots are the simplest case), the query stream is decoupled from the host's token stream: the write sources are designated backbone features of the current frame, while the query stream alone evolves across stacked memory layers. The readout then enters the policy as a separate representation—attention prefix slots, appended memory tokens, or the pre-head representation—leaving the host's own token stream untouched.

Our $\pi_{0.5}$ host follows this pattern. The frozen VLM backbone has 18 transformer layers, and DRAM layers are aligned with them: memory layer ℓ reads and writes using features of backbone layer ℓ, so historical context is exchanged at matching representation depths. The query stream is a set of M learned query slots, and the write sources X_t^K, X_t^V are tapped from the backbone in one of three configurations, illustrated in Figure 3: (i) *Native KV* reuses the backbone attention's own key–value projections of each layer as the write sources, tapping representations the backbone already computes; (ii) *Hidden tokens* tap the hidden token states of each backbone layer and projects them with DRAM's own $W_K^{(h)}, W_V^{(h)}$; (iii) *Last hidden* uses the final-layer VLM hidden tokens as a shared write source across memory layers, allowing the memory bank depth to be chosen independently of the VLM depth; only the final memory readout is passed to the action expert. Section 4.3 compares the three configurations empirically (Table 3).

For the readout, the query slots retrieve from each memory layer, and the fused readouts are injected as prefix slots into the action expert's attention. The prefix slots use a reserved RoPE index (-1), for which rotation of their keys is skipped; the action expert's original tokens retain their position indices and thus their relative positions. The zero-initialized output projection makes FUSE an identity at initialization, so the memory readout initially adds no residual change to the query representations.

B.2 Shared-source integration: the DiT host

Without a deep multimodal backbone to tap, the current frame's own tokens serve as the query, key, and value sources at once, and each memory layer returns the fused sequence $Y_t^{(\ell)}$ in place of the

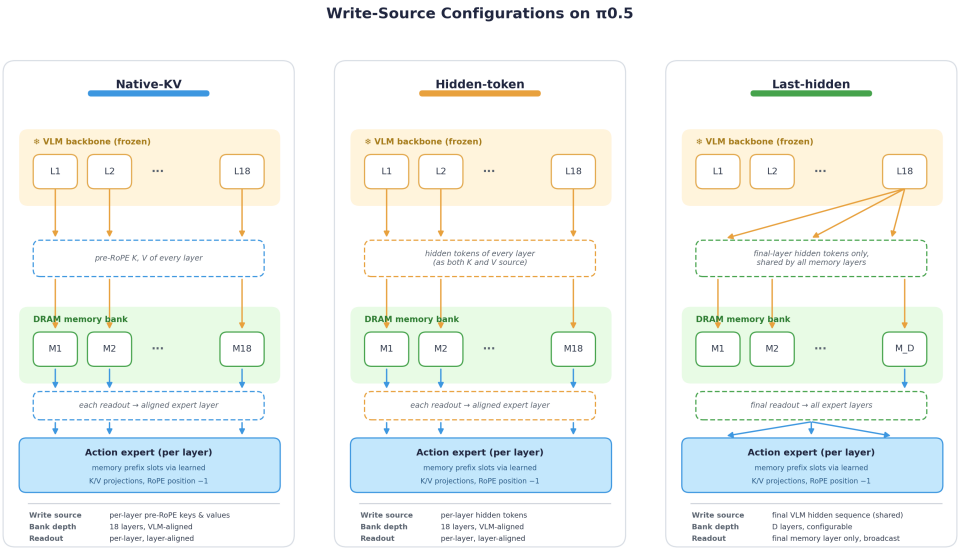

Figure 3: Write-source configurations for the $\pi_{0.5}$ host (separate-source integration). DRAM layers are aligned with the VLM backbone layers, and the write sources of memory layer ℓ are tapped from (i) the backbone attention's native key–value projections (*Native KV*), (ii) intermediate hidden tokens (*Hidden tokens*), or (iii) only the final-layer hidden states (*Last hidden*). Query slots, readout injection, and the memory update are identical across configurations.

raw readout, feeding layer $\ell{+}1$ as in an ordinary transformer. Since history is mixed into the tokens while preserving their structure, the action head consumes Y_t through its original interface with no modification; one simply designates which layer's output feeds it.

Our DiT host follows this pattern, in the style of GR00T N1.7: a vision tower encodes all camera views of the current frame at once, and the resulting patch tokens are flattened into a single sequence on which a DiT action head conditions through cross-attention. The frame tokens serve as the query, key, and value sources of DRAM ($M{=}N$); each memory layer applies the full read–fuse–write procedure, optionally followed by the intra-frame mixer, and memory layers stack with a depth chosen independently of the host. Because history is carried by the tokens themselves—preserving their number, ordering, and positional encoding—the DiT head reads the fused sequence through its original cross-attention interface unchanged.

Practical principles. Two guidelines apply across both patterns. First, history-carrying tokens should not disturb the host's positional scheme: injected readouts are marked as positionless (a reserved RoPE index with unrotated values, as above) or delivered through channels that bypass positional encoding altogether. Second, readout granularity is orthogonal to the write-source choice: layer-aligned readouts suit deep hosts whose representations evolve across layers, whereas a single final-layer readout suffices for shallow hosts or single-layer memory.

C Derivation and Properties of the Frame-Level Joint Write

This appendix derives the closed form of the frame-synchronous joint write (Section 3.3), proves the properties stated in the main text, and details its numerical realization, including gradient flow through the solve. Throughout, we suppress the frame index t and the head index h: $K \in \mathbb{R}^{N\times d_k}$ and $V \in \mathbb{R}^{N\times d_v}$ stack the unit-normalized keys and the values of the N write tokens of one frame row-wise, $k_i \in \mathbb{R}^{1\times d_k}$ and $v_i \in \mathbb{R}^{1\times d_v}$ denote their i-th rows (so the prediction for token i is the row vector $k_i S$), $W = \mathrm{diag}(w_1, \ldots, w_N)$ collects the non-negative write weights, and $\bar{S} = \gamma_t S_{t-1}$ is the retained state.

C.1 NORMAL EQUATION, UNIQUENESS, AND CLOSED FORMS

Design of the joint objective. The single-token delta rule takes one *explicit* gradient step on the local prediction error $\mathcal{L}_i(S) = \frac{1}{2}\|k_i S - v_i\|_2^2$, evaluated at the retained state: $S = \bar{S} - \beta\,\nabla\mathcal{L}_i(\bar{S})$. Applying it to the N tokens of a frame one after another chains N such steps, each evaluated at a different intermediate state, so the result depends on the within-frame order: already with a scalar state, γ=1, β=1 and the two tokens $(1,1)$ and $(1,0)$, sequential writes end at the *last* token's value (0 or 1 depending on the order), whereas fitting both tokens jointly yields $\frac{1}{2}$ irrespective of their order. We therefore evaluate all per-token errors against a *single* candidate state and minimize them jointly,

$$\mathcal{L}(S) \;=\; \frac{1}{2}\sum_{i=1}^{N} w_i\,\|k_i S - v_i\|_2^2, \tag{19}$$

which asks one state to explain all current key–value pairs simultaneously. By itself, however, $\mathcal{L}$ says nothing about the directions of the state space that the current keys do not span: when $N < d_k$ the minimizer is not unique, and the history stored in those directions could be altered arbitrarily. We therefore add an anchor term and define the joint write as the proximal step,

$$S \;=\; \arg\min_{S\in\mathbb{R}^{d_k\times d_v}} \;\frac{1}{2}\|S-\bar{S}\|_F^2 \;+\; \mathcal{L}(S), \tag{20}$$

This objective balances fitting the current key–value associations against staying close to $\bar{S}$. Components not addressed by the current keys remain unchanged, while addressed components are updated according to the strength of the current evidence. The anchor also fixes the relative weighting—with its weight set to 1, each w_i plays the role of a per-token step size, and the odds choice $w_i = \hat{\beta}_i/(1-\hat{\beta}_i)$ makes the single-token reduction reproduce the gated delta rule with strength exactly $\hat{\beta}_i$ (Appendix C.2).

From sum to matrix form. We write the objective of Equation (20)—identical to Equation (15) in Section 3.3—more explicitly:

$$\begin{aligned} J(S) &= \frac{1}{2}\|S-\bar{S}\|_F^2 + \frac{1}{2}\sum_{i=1}^{N} w_i\|k_i S - v_i\|_2^2 \\ &= \frac{1}{2}\|S-\bar{S}\|_F^2 \;+\; \frac{1}{2}\left\|\begin{bmatrix}\sqrt{w_1}\,(k_1 S - v_1)\\ \vdots \\ \sqrt{w_N}\,(k_N S - v_N)\end{bmatrix}\right\|_F^2 \\ &= \frac{1}{2}\|S-\bar{S}\|_F^2 \;+\; \frac{1}{2}\big\|W^{1/2}(KS-V)\big\|_F^2. \end{aligned} \tag{21}$$

By the definition of the Frobenius norm: the i-th row of $KS-V$ is the row vector $k_i S - v_i$, left-multiplying by $W^{1/2} = \mathrm{diag}(\sqrt{w_1},\ldots,\sqrt{w_N})$ scales row i by $\sqrt{w_i}$, and the squared Frobenius norm sums the per-row squared norms. The matrix form of the second term is thus simply a compact notation for the weighted sum of per-token prediction errors.

Gradient and normal equation. Since W is symmetric, the objective expands as

$$\begin{aligned} J(S) &= \frac{1}{2}\|S-\bar{S}\|_F^2 \;+\; \frac{1}{2}\big\|W^{1/2}(KS-V)\big\|_F^2 \\ &= \frac{1}{2}\,\mathrm{tr}\left((S-\bar{S})^\top(S-\bar{S})\right) \;+\; \frac{1}{2}\,\mathrm{tr}\left((KS-V)^\top W\,(KS-V)\right). \end{aligned} \tag{22}$$

To locate the minimizer, perturb S along an arbitrary direction $\Delta S \in \mathbb{R}^{d_k\times d_v}$ and expand in ε, using $\frac{1}{2}\|U+\varepsilon D\|_F^2 = \frac{1}{2}\|U\|_F^2 + \varepsilon\langle U, D\rangle_F + \frac{\varepsilon^2}{2}\|D\|_F^2$ on both terms:

$$\begin{aligned} J(S+\varepsilon\Delta S) &= J(S) \;+\; \varepsilon\left[\,\big\langle S-\bar{S},\,\Delta S\big\rangle_F \;+\; \big\langle W^{1/2}(KS-V),\,W^{1/2}K\Delta S\big\rangle_F\,\right] \\ &\quad+\; \frac{\varepsilon^2}{2}\left[\,\|\Delta S\|_F^2 \;+\; \big\|W^{1/2}K\Delta S\big\|_F^2\,\right] \\ &= J(S) \;+\; \varepsilon\,\big\langle (S-\bar{S}) + K^\top W(KS-V),\;\Delta S\big\rangle_F \;+\; \frac{\varepsilon^2}{2}\,\big\langle \Delta S,\;A\,\Delta S\big\rangle_F, \end{aligned} \tag{23}$$

where $\langle\cdot\rangle_F$ is the Frobenius inner product and $A \triangleq I_{d_k} + K^\top W K$. The second line of Equation (23) moves $W^{1/2}K$ across the inner product by the adjoint rule $\langle X, W^{1/2}K\Delta S\rangle_F = \langle K^\top W^{1/2}X, \Delta S\rangle_F$ (valid since W is symmetric), and collects the quadratic terms via $\|W^{1/2}K\Delta S\|_F^2 = \langle \Delta S, K^\top W K \Delta S\rangle_F$. Since the first-order coefficient holds for every ΔS, it identifies the gradient,

$$\nabla_S J(S) \;=\; (S - \bar{S}) \;+\; K^\top W(KS - V) \;\in\; \mathbb{R}^{d_k \times d_v}, \tag{24}$$

an affine map of S: the anchor contributes $S - \bar{S}$ and each token contributes $w_i\, k_i^\top (k_i S - v_i)$, its prediction residual pulled back by its key. Setting $\nabla_S J(S) = 0$ yields the normal equation

$$\underbrace{\left(I_{d_k} + K^\top W K\right)}_{A} S \;=\; \bar{S} + K^\top W V. \tag{25}$$

The second-order coefficient of Equation (23) is the Hessian quadratic form, and it satisfies

$$\langle \Delta S,\, A\,\Delta S\rangle_F \;=\; \|\Delta S\|_F^2 + \left\|W^{1/2}K\Delta S\right\|_F^2 \;\geq\; \|\Delta S\|_F^2 \;>\; 0 \qquad \text{for all } \Delta S \neq 0, \tag{26}$$

because the Gram matrix $K^\top W K = \sum_{i=1}^N w_i\, k_i^\top k_i$ is positive semidefinite for any non-negative weights. Hence $A \succeq I_{d_k} \succ 0$ for *any* keys—even repeated or rank-deficient ones—J is strictly convex, and the solution of Equation (25) is the unique global minimizer. The identity supplied by the anchor is essential here: without it, the fit term alone has no unique solution whenever the current keys fail to span $\mathbb{R}^{d_k}$.

Closed forms. Since $A \succeq I_{d_k}$, it is invertible, and left-multiplying Equation (25) by A^{-1} gives the aggregate closed form. The correction form follows from rewriting the retained state so that A^{-1} cancels: by the definition $A \triangleq I_{d_k} + K^\top W K$,

$$A\bar{S} \;=\; (I_{d_k} + K^\top W K)\,\bar{S} \;=\; \bar{S} + K^\top W K \bar{S}, \qquad \text{i.e.,} \qquad \bar{S} \;=\; A\bar{S} - K^\top W K \bar{S}. \tag{27}$$

Substituting this expression for $\bar{S}$ into the aggregate form lets A^{-1} act on $A\bar{S}$ and recover a bare $\bar{S}$:

$$\begin{aligned} S &= A^{-1}\left(\bar{S} + K^\top W V\right) && \text{(aggregate form)} \\ &= A^{-1}\left(A\bar{S} - K^\top W K \bar{S} + K^\top W V\right) && \text{(substituting Equation (27))} \\ &= \bar{S} + A^{-1}K^\top W\left(V - K\bar{S}\right) \;=:\; \bar{S} + \Delta. && \text{(correction form)} \end{aligned} \tag{28}$$

The aggregate form exposes how retained history and new evidence are blended by A^{-1}; the correction form writes the update as the retained state plus a coupled correction Δ on the prediction residual $V - K\bar{S}$, and is the one we compute in practice (Appendix C.4). Equivalently, $\Delta = -[\nabla^2 J]^{-1}\nabla J(\bar{S})$ is the single Newton step from $\bar{S}$ on the quadratic J, which lands exactly at the minimizer.

C.2 Single-token correspondence

For a single association (k, v) with unit-normalized key $\|k\|_2 = 1$ and weight w, the system matrix is a rank-one update of the identity, $A = I_{d_k} + w\, k^\top k$, and the Sherman–Morrison formula gives

$$A^{-1} \;=\; I_{d_k} - \frac{w}{1 + w\, kk^\top}\, k^\top k \;=\; I_{d_k} - \frac{w}{1+w}\, k^\top k, \tag{29}$$

since $kk^\top = \|k\|_2^2 = 1$. Applying this to the aggregate closed form,

$$\begin{aligned} S = A^{-1}\left(\bar{S} + w\, k^\top v\right) &= \bar{S} - \frac{w}{1+w}\, k^\top (k\bar{S}) + w\, k^\top v - \frac{w^2}{1+w}\, k^\top (kk^\top) v \\ &= \bar{S} + \frac{w}{1+w}\, k^\top \left(v - k\bar{S}\right). \end{aligned} \tag{30}$$

The odds reparameterization $w = \hat{\beta}/(1-\hat{\beta})$ of Equation (14) is chosen precisely so that the effective coefficient collapses to the gate itself:

$$\frac{w}{1+w} \;=\; \hat{\beta} \qquad \Longrightarrow \qquad S \;=\; \bar{S} + \hat{\beta}\, k^\top\left(v - k\bar{S}\right) \;=\; \gamma_t S_{t-1} + \hat{\beta}\, k^\top\left(v - \gamma_t k S_{t-1}\right), \tag{31}$$

which is exactly the gated delta rule of Equation (4) with correction strength $\hat{\beta}$. The frame-synchronous joint write therefore strictly generalizes the gated delta rule: the latter is the $N = 1$ special case of the former. The same equivalence between an odds-parameterized gate and the rank-one closed form is derived concurrently in the analysis of Zhu (2026). The cap $\hat{\beta} \leq 0.99$ bounds $w \leq 99$, keeping A well conditioned (Appendix C.4).

C.3 Permutation invariance and bounded accumulation

Permutation invariance (Property ii). Let $\Pi \in \{0,1\}^{N\times N}$ be a permutation matrix and consider the permuted inputs $K' = \Pi K$, $V' = \Pi V$, and $W' = \Pi W \Pi^\top$ (the diagonal entries of W, including the padding mask carried by $w_i = p_i\hat{\beta}_i/(1-\hat{\beta}_i)$, are permuted along with the rows). Both sides of the normal equation are unchanged:

$$K'^\top W' K' = K^\top \Pi^\top \Pi W \Pi^\top \Pi K = K^\top W K, \qquad K'^\top W' V' = K^\top W V, \tag{32}$$

since $\Pi^\top\Pi = I_N$. Hence $A' = A$; the right-hand side is identical, and the unique minimizer S is the same in exact arithmetic. No ordering of the tokens within a frame—spatial raster, camera identity, or language order—can influence the written state.

Bounded evidence accumulation (Property iii). Suppose one association (k, v) with $\|k\|_2 = 1$ and weight w appears n times in a frame. The Gram matrix is $K^\top W K = nw\, k^\top k$, and Sherman–Morrison as in Equation (29)–Equation (30) gives

$$S \;=\; \bar{S} + \frac{nw}{1+nw}\, k^\top \big(v - k\bar{S}\big). \tag{33}$$

The effective correction strength $nw/(1+nw)$ increases strictly with n for $w > 0$, approaching 1 from below. Repeated evidence therefore *reinforces* the write—unlike token-count averaging, which would keep the strength at $w/(1+w)$ regardless of n—while the correction strength remains below 1 for every finite n.

C.4 Numerical realization: masking, precision, and gradient flow

Masking. Padded write tokens carry $p_i = 0$, hence $w_i = 0$: they contribute neither to the Gram matrix nor to the right-hand side, and the mask-aware statistics of the gates ($\bar{g}_t$) use the same mask. Fully padded frames skip the update entirely, leaving S_{t-1} untouched (in particular, no decay is applied); valid frames whose write weights are all zero give $A = I_{d_k}$ and return only the decayed state $S_t = \bar{S}$.

Conditioning and the role of the cap. With unit-normalized keys, $\lambda_{\max}(K^\top W K) \le \sum_{i=1}^N w_i \|k_i\|_2^2 = \sum_{i=1}^N w_i \le 99\,N$, while the anchor guarantees $\lambda_{\min}(A) \ge 1$. Hence

$$\kappa(A) \;\le\; 1 + 99\,N, \tag{34}$$

linear in the number of write tokens and independent of the data. The cap $\hat{\beta} \le 0.99$ is what makes this bound possible, and the identity from the anchor term is what makes the lower bound data-independent—the numerical counterpart of its uniqueness role in Appendix C.1.

Forward solve. We never materialize A^{-1}. Writing $S = \bar{S} + \Delta$, the normal equation becomes $A\,\Delta = K^\top W\big(V - K\bar{S}\big)$, which we solve by Cholesky factorization $A = LL^\top$ per frame, head, and memory layer; the same factorization is reused for all d_v columns of the right-hand side. The Gram matrix and the solve are computed in FP32 with autocast and TF32 disabled, and the persistent states are stored in FP32; the rest of the network is unaffected. The per-frame cost is $(O(Nd_k^2 + Nd_kd_v + d_k^3 + d_k^2 d_v)$ per head, independent of episode length.

Gradient flow. The solve is differentiable, and we backpropagate through it by implicit differentiation rather than through the Cholesky operator. Given the upstream gradient $G = \partial\mathcal{L}/\partial S \in \mathbb{R}^{d_k\times d_v}$ at the converged solution, the adjoint variable Λ solves the transposed system—which is the same system, since A is symmetric—reusing the stored factorization:

$$A\,\Lambda = G. \tag{35}$$

The input gradients then follow from differentiating $AS = \bar{S} + K^\top W V$:

$$\begin{aligned}
&\frac{\partial\mathcal{L}}{\partial\bar{S}} = \Lambda, \qquad \frac{\partial\mathcal{L}}{\partial V} = WK\Lambda, \qquad \frac{\partial\mathcal{L}}{\partial w_i} = \big(K\Lambda\big)_i\big(V - KS\big)_i^\top, \\
&\frac{\partial\mathcal{L}}{\partial K} = W\big(V - KS\big)\Lambda^\top \;-\; WK\,\Lambda S^\top,
\end{aligned} \tag{36}$$

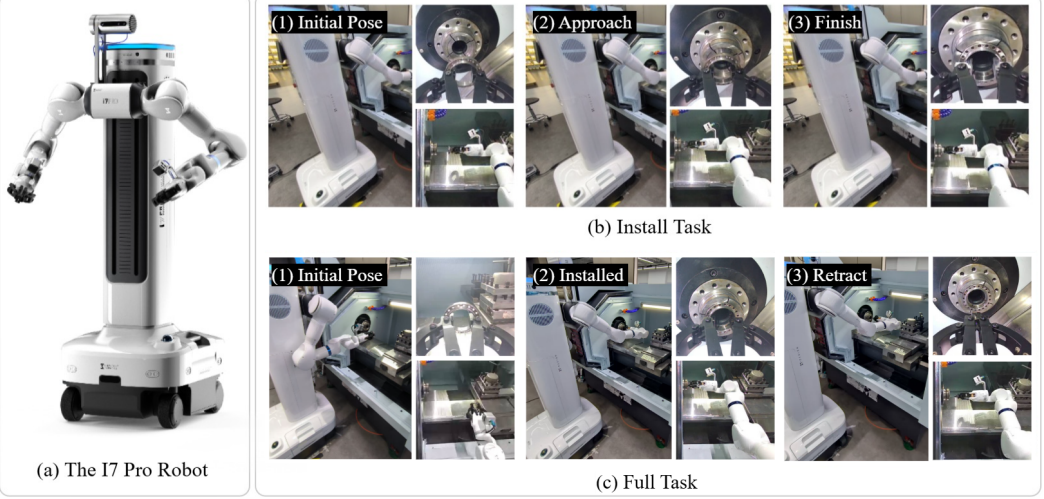

Figure 4: Real-world experimental platform and manipulation tasks. (a) The I7 Pro robot. (b) The Install task, which requires precise alignment and insertion from a predefined pose near the target. (c) The Full task, which extends Install with an initial approach, workpiece release, and arm retraction to complete the loading sequence. For each task, three representative execution stages are illustrated using a global view, a close-up view of the manipulation, and a wrist-camera observation.

where $(\cdot)_i$ denotes the i-th row. The weight gradient has a simple interpretation: it is the inner product between what token i retrieves from the update, $(K\Lambda)_i$, and its residual at the solution, $(V - KS)_i$. From $\partial\mathcal{L}/\partial\bar{S}$, gradients flow further to the retention gate via $\partial\mathcal{L}/\partial\gamma_t = \langle\Lambda, S_{t-1}\rangle_F$ and to the previous state via $\gamma_t\Lambda$, continuing the recurrent chain across the TBPTT window; the readout path contributes its own gradient to S_{t-1} through Equation (11). Gradients with respect to w_i and the gate pre-activations follow from $w_i = p_i\hat{\beta}_i/(1-\hat{\beta}_i)$ by the chain rule. In practice we implement the forward and backward solves inside a custom autograd function that stores L from the forward pass, so that Equation (35) costs one triangular solve pair rather than a refactorization.

D Details of Real-World Experiments

We conduct the real-world evaluation on the I7 Pro, a mobile manipulation platform equipped with dual robotic arms and onboard visual sensing, as shown in Figure 4. We consider two manipulation tasks in a CNC workspace: *Install* and *Full*. In both tasks, the robot starts with the workpiece already grasped. In the Install task, the arm begins at a predefined pose near the installation target and performs precise alignment and insertion. The Full task extends this procedure into a complete loading sequence: the arm starts from a more distant initial pose, moves to the starting pose of the Install task, and completes the same alignment and insertion procedure. After insertion, the robot opens the gripper to release the workpiece and retracts the arm. Thus, Install focuses on fine manipulation near the target, while Full additionally requires the initial approach, workpiece release, and arm retraction. For each task and method, we conduct 20 evaluation trials and report the success rate. Representative execution stages and corresponding close-up views are shown in Figure 4.

E Limitations

Our evaluation considers episodic manipulation, with memory reset at episode boundaries. Memory management during continuous execution across successive tasks remains unexplored. Future work could investigate when to retain or reset memory as task instructions change.