STAM: SPATIO-TEMPORAL ADAPTIVE COARSE-FINE VISUAL MEMORY FOR GENERALIST ROBOT POLICIES

Anonymous authors
Paper under double-blind review

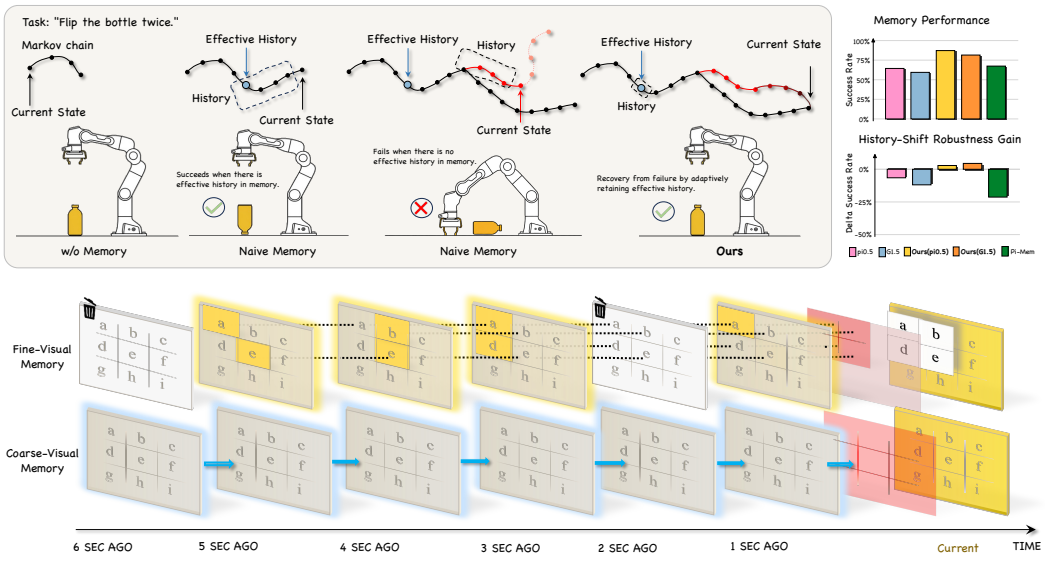


Figure 1: **Overview of STAM.** STAM is a plug-in visual memory framework for general robot policies. **Top left:** fixed-horizon history policies can solve non-Markovian tasks, but they can also overfit to out-of-distribution noisy histories (red lines); STAM uses spatio-temporal adaptive memory to pinpoint task-relevant historical evidence and avoid irrelevant-history overfitting. **Top right:** STAM augments general robot policies with memory, improving memory-required manipulation while preserving robustness when history is unnecessary. **Bottom:** illustration of spatio-temporal adaptive fine visual memory (upper), which retrieves patch-level historical evidence at selected recall times, and adaptive coarse visual memory (lower), which provides compressed global memory.

ABSTRACT

Generalist robot policies need historical visual evidence when the current observation is insufficient for action selection. However, compressing history can discard useful details, while indiscriminate history conditioning can introduce irrelevant information. We propose **STAM**, a plug-in spatio-temporal adaptive coarse-fine visual memory module for generalist robot policies. Its coarse-fine visual memory combines compressed historical context with directly retrievable visual details. A spatio-temporal recall map determines which image patches access memory and how far back to retrieve fine visual evidence. Layerwise coarse-fine memory fusion integrates both memory sources into the selected patches without modifying the downstream policy backbone. An action-guided three-stage training procedure learns recall decisions from action errors without manual recall-map annotations. Experiments across four simulation benchmarks and four real-world robot platforms demonstrate improved memory-dependent manipulation while maintaining strong general manipulation performance. On MIKASA-Robo, STAM improves average success by 33.2 and 28.0 percentage points over the original $\pi_{0.5}$ and GR00T N1.5 policies, respectively. Code will be available after acceptance.

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.

1 INTRODUCTION

Generalist robot policies built under the Markov assumption have made substantial progress in controlling robots in the physical world. However, most real-world robotic manipulation tasks are non-Markovian (Shi et al., 2026; Gao et al., 2026). This means that the current observation alone may be insufficient to determine the correct action. Examples include wiping a table three times or pressing a button twice. Recent methods therefore augment robot policies with multiple past observations or compressed history representations (Li et al., 2025; Shi et al., 2026; Torne et al., 2026; Chung et al., 2025). Yet naively conditioning a policy on more history does not guarantee reliable memory. Irrelevant or noisy historical observations can distract the policy from current evidence and reduce its robustness to distribution shifts (Gao et al., 2026). This raises a key question: **How can generalist robot policies preserve and selectively recall historical visual evidence?**

Reliable visual memory must preserve historical visual evidence and selectively access it. First, memory must preserve fine-grained visual evidence over long horizons. Existing methods often compress history into compact representations for efficiency (Li et al., 2026; Jang et al., 2025). However, this compression may discard details needed for later actions, such as an object's appearance and location. Second, access to the retained history must be selective. Useful evidence may be localized in space and distant in time, while other historical observations may be irrelevant or noisy. Therefore, generalist robot policies should access memory selectively in both space and time.

To achieve these goals, we propose **STAM**, a plug-in spatio-temporal adaptive coarse-fine visual memory framework for generalist robot policies. It augments the vision encoder without modifying the downstream policy backbone. STAM uses a spatio-temporal recall map to control access to coarse-fine visual memory. **(i) Spatio-temporal recall map.** Spatially, the recall map determines which image patches can access memory. Temporally, it assigns each selected patch a continuous recall time that specifies how far back to retrieve visual evidence. **(ii) Coarse-fine visual memory.** A state-space memory core (SSMC) produces coarse memory as a compressed summary of past observations. Fine memory is retrieved from an episodic patch buffer that preserves local historical visual details. For the patches selected by the recall map, layerwise coarse-fine memory fusion injects fine memory into shallow vision layers and coarse memory into deeper layers. Training first adapts the policy to memory inputs using randomly sampled recall maps. We then use action errors to search for recall-map pseudo-labels. These pseudo-labels guide teacher-forced joint training of the memory components and the policy. In summary, our contributions are as follows:

- We propose **STAM**, a plug-in visual memory module for generalist robot policies. It augments the vision encoder with historical visual evidence without modifying the downstream policy backbone.
- We design **coarse-fine visual memory** that combines compressed historical context with directly retrievable visual details. Layerwise coarse-fine memory fusion injects these complementary signals into the selected image patches.
- We introduce a **spatio-temporal recall map** that selects which image patches access memory and predicts how far back to retrieve visual evidence. An action-guided training procedure learns these recall decisions from action errors.
- Comprehensive experiments across simulation and real-world tasks demonstrate that STAM improves non-Markovian manipulation across different backbones while maintaining strong general manipulation performance.

2 RELATED WORKS

2.1 VISION-LANGUAGE-ACTION MODELS

Vision-language-action (VLA) models predict robot actions from visual observations and language instructions. RT-1 and RT-2 scale this paradigm through large robot datasets and web-scale vision-language pretraining (Brohan et al., 2022; 2023). Octo and OpenVLA support generalist control and downstream adaptation (Ghosh et al., 2024; Kim et al., 2024; 2025). Further advances improve action generation through continuous action heads, flow matching, and efficient tokenization (Black et al., 2024; Physical Intelligence et al., 2025; Pertsch et al., 2025; NVIDIA GEAR, 2025), as well as spatial grounding (Qu et al., 2025). These policies primarily use current observations or short

history windows. STAM extends their access to historical visual evidence through the vision encoder without modifying the downstream policy backbone.

2.2 General robot policies with memory

Memory-augmented robot policies address non-Markovian manipulation using historical observations or learned memory states. CronusVLA and ContextVLA encode history as feature queues or context tokens (Li et al., 2025; Jang et al., 2025), while MEM combines visual and semantic memory (Torne et al., 2026). MemoryVLA, ReMem-VLA, and HAMLET use memory banks, recurrent queries, or learned moment tokens (Shi et al., 2026; Li et al., 2026; Koo et al., 2026). Embodied-SlotSSM maintains object-centric memory states (Chung et al., 2025). These approaches often rely on compressed history or fixed multi-frame context. Gated Memory Policy learns when to use history from action errors (Gao et al., 2026), while past-token prediction adds history supervision to diffusion policies (Torne et al., 2025). STAM combines compressed coarse memory with retrievable fine visual evidence. Its spatio-temporal recall map is trained with action-guided supervision to select which patches access memory and how far back to retrieve visual evidence.

2.3 State space models

State space models (SSMs) efficiently capture long-range dependencies. S4 and S5 support parallel training and linear-time sequence inference (Gu et al., 2022; Smith et al., 2023), while H3 motivates hybrid attention-SSM designs to improve recall and comparison (Fu et al., 2023). Mamba introduces input-conditioned state updates, and Mamba-2 connects selective SSMs to structured attention (Gu & Dao, 2023; Dao & Gu, 2024). Vision Mamba, VMamba, and Mamba-ND extend selective scans to visual and multidimensional data (Zhu et al., 2024; Liu et al., 2024; Li et al., 2024). Related work selects informative tokens for long-form video (Wang et al., 2023) or maintains modular slots to separate entities (Jiang et al., 2024). STAM uses an SSM for compressed coarse memory and an episodic patch buffer to preserve fine visual evidence that state compression may discard.

3 Method

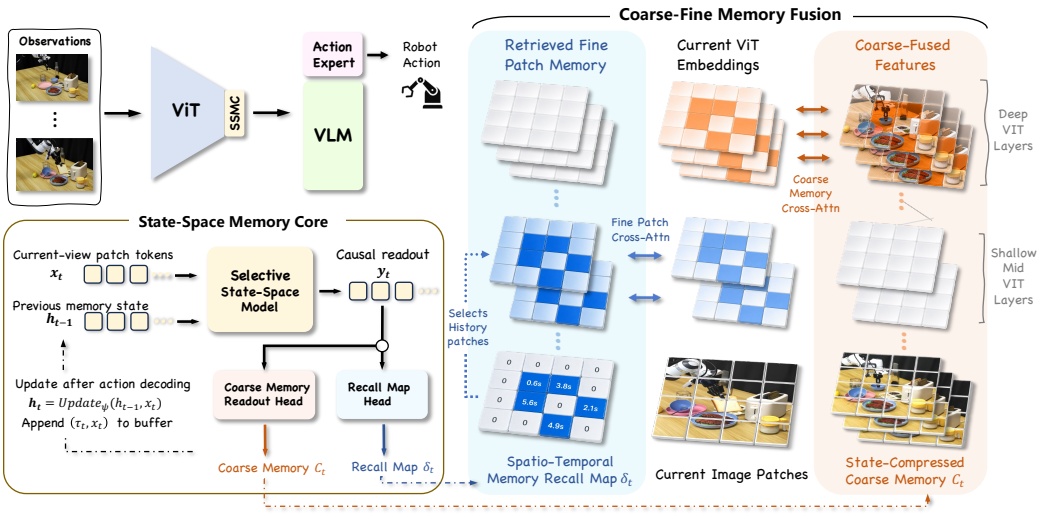


Figure 2: **Overview of STAM. Left:** The state-space memory core (SSMC) reads the previous memory state to produce coarse memory and a spatio-temporal recall map.The memory state and episodic patch buffer are updated after action decoding. **Right:** The spatio-temporal recall map selects which patches access memory and how far back to retrieve historical visual evidence. Layerwise coarse-fine memory fusion injects retrieved fine patch memory into shallow ViT layers and state-compressed coarse memory into deeper layers, both restricted to the selected patches.

In this section, we describe STAM and its training procedure. As shown in Fig. 2, STAM augments the vision encoder without modifying the downstream policy backbone. We first introduce coarse-fine visual memory in Sec. 3.1 to retain compressed historical context and fine visual details. Next, Sec. 3.2 presents the spatio-temporal recall map, which controls which patches access this memory and how far back they recall. Finally, we describe the action-guided training procedure in Sec. 3.3.

3.1 COARSE-FINE VISUAL MEMORY

To retain both historical context and fine visual details, we maintain two complementary memory representations. At decision step t, the vision encoder extracts shallow view-patch features x_t from the multi-view observation I_t. The state-space memory core (SSMC) compresses these features over time to provide coarse visual memory. An episodic patch buffer stores timestamped features for retrieving fine visual memory.

Coarse visual memory. The SSMC maintains a recurrent history state $h_t[p]$ for each view-patch index $p \in \{1, \dots, P\}$, where P is the total number of tokens after flattening views and image patches. At decision step t, the SSMC first reads a causal memory readout y_t from the previous state h_{t-1} and maps y_t to state-compressed coarse memory c_t. The current view-patch tokens x_t are used to update the state only after action decoding, so the memory used by the current decision is read from history.

$$\begin{aligned} y_t &= \mathrm{Read}_\psi(h_{t-1}), \\ c_t &= W_c \mathrm{LN}(y_t), \\ h_t &= \mathrm{Update}_\psi(h_{t-1}, x_t) \quad \text{after action decoding.} \end{aligned} \tag{1}$$

Here ψ denotes the SSMC parameters, W_c is the coarse memory readout projection, and LN denotes layer normalization. The recall map head uses the same causal readout as its input for patch-level memory control.

Fine visual memory. The episodic patch buffer stores fine-grained visual memory as timestamped view-patch tokens from the current episode. After the action at step i has been decoded, STAM appends the pair (τ_i, x_i) to the buffer; therefore the buffer available at step t contains only historical entries, written compactly as $\mathcal{B}_{t-1} = \{(\tau_i, x_i)\}_{i<t}$. Here τ_i is the physical timestamp of step i, and x_i denotes the corresponding view-patch tokens. Fine patch memory is later read from this buffer according to the recall map. Because the stored entries remain view-patch tokens rather than SSM-compressed states, the buffer preserves local historical visual evidence for later retrieval.

Together, coarse and fine visual memory retain historical context and local visual details. We integrate these memories into the vision encoder through layerwise coarse-fine memory fusion. Fine patch memory enters shallow ViT layers, while state-compressed coarse memory enters deeper layers. Both paths are restricted to patches selected by the spatio-temporal recall map detailed in Sec. 3.2. Layer assignments are detailed in Appendix A.2.

3.2 SPATIO-TEMPORAL RECALL MAP

To selectively access coarse-fine visual memory, we predict a spatio-temporal recall map. Each entry controls whether a view-patch uses memory and how far back it recalls. The recall map head predicts these values from the causal readout y_t:

$$\begin{aligned} \hat{\delta}_t &= (\tau_t - \tau_0)\mathrm{HardSigmoid}(W_\delta \mathrm{LN}(y_t)), \\ \mathcal{A}_t &= \{p : \hat{\delta}_t[p] > \epsilon_\delta\}, \qquad \delta_t[p] = \hat{\delta}_t[p]\, \mathbf{1}[p \in \mathcal{A}_t]. \end{aligned} \tag{2}$$

Here W_δ is the recall map head projection, τ_t is the current timestamp, and τ_0 is the episode start time. The HardSigmoid function maps predictions to $[0, 1]$, which scales the recall time to the elapsed episode duration. The threshold ϵ_δ determines the active patch set $\mathcal{A}_t$, and $\mathbf{1}[\cdot]$ is the indicator function. Spatial selection is determined by $\mathcal{A}_t$: inactive patches have $\delta_t[p] = 0$ and receive neither fine nor coarse memory. For active patches, $\delta_t[p]$ specifies the temporal look-back duration for fine-memory retrieval. If the historical buffer is empty, we set $\mathcal{A}_t = \emptyset$ and $\delta_t = 0$.

For each active patch, the target timestamp is $s_t[p] = \tau_t - \delta_t[p]$. We retrieve the latest buffered frame at or before this target, falling back to the earliest frame if the target precedes the buffer. For

a nonempty buffer whose first timestamp is τ_1, the lookup is

$$i_t^\star[p] = \max\{i < t : \tau_i \le \max(\tau_1, s_t[p])\}, \quad p \in \mathcal{A}_t,$$
$$f_t[p] = \begin{cases} x_{i_t^\star[p]}[p], & p \in \mathcal{A}_t, \\ \varnothing, & p \notin \mathcal{A}_t. \end{cases} \tag{3}$$

Here $f_t[p]$ is the retrieved fine patch feature, and $\varnothing$ denotes no retrieved token. Nearest-older lookup selects a stored frame for each active patch and retrieves its feature at the same view-patch index without interpolation. During fusion, these active patches also receive coarse memory read from the compressed history state. The recall map therefore controls access to both memory sources, while its predicted recall times determine only which fine patch features are retrieved. Additional buffer details are provided in Appendix A.4.

3.3 Training Pipeline

In this section, we present the training pipeline of STAM, as shown in Algorithm 1. Demonstrations provide action targets but no recall-map annotations. We therefore adopt a three-stage training procedure. The three stages adapt the policy to memory inputs, generate recall-map pseudo-labels from action errors, and jointly train recall prediction and action generation, respectively.

Algorithm 1 Training Pipeline for STAM.

Require: training split $\mathcal{D}_{\text{train}}$, recall-label split $\mathcal{D}_{\text{val}}$, search budget N

Stage 1: Random-Recall Memory Adaptation
- Freeze the recall map head.
- **for** minibatch from $\mathcal{D}_{\text{train}}$ **do**
  - Sample δ_t^{rand} and run causal STAM forward.
  - Update trainable policy and memory parameters with $\mathcal{L}_{\text{action}}$.
- **end for**

Stage 2: Action-Guided Recall Label Search
- Freeze the policy; initialize $\mathcal{D}_{\text{recall}} \leftarrow \emptyset$.
- **for** step t in $\mathcal{D}_{\text{val}}$ **do**
  - $\mathcal{C}_t \leftarrow \{0\} \cup \text{SampleMaps}(N-1)$.
  - $E_t(\delta) \leftarrow \mathcal{D}(\pi_\theta(o_t; \delta), a_t)$ for each $\delta \in \mathcal{C}_t$.
  - $\delta_t^\star \leftarrow \arg\min_{\delta \in \mathcal{C}_t} E_t(\delta)$.
  - **if** $E_t(0) - E_t(\delta_t^\star) \le \epsilon$ **then** $\delta_t^\star \leftarrow 0$.
  - Compute confidence w_t and store $(x_t, y_t, \delta_t^\star, w_t)$ in $\mathcal{D}_{\text{recall}}$.
- **end for**

Stage 3: Teacher-Forced Memory Joint Training
- Unfreeze trainable STAM modules and selected VLA parameters.
- **for** joint-training step using $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{recall}}$ **do**
  - Predict $\hat{\delta}_t$; select $\delta_t^{\text{train}} \leftarrow \begin{cases} \delta_t^\star, & \text{teacher forcing,} \\ \hat{\delta}_t, & \text{scheduled sampling.} \end{cases}$
  - $\hat{a}_t \leftarrow \pi_\theta(o_t; \delta_t^{\text{train}})$; update with $\mathcal{L}_{\text{stage3}}$ in Eq. (5).
- **end for**
- **return** trained STAM-augmented policy π_θ

Stage 1: Random-recall memory adaptation. We first train the policy with sampled recall maps under the original action loss. This allows the policy to use historical visual evidence before learning where and when to recall it, providing a basis for evaluating memory choices through action errors.

Stage 2: Action-guided recall label search. We freeze the adapted policy and evaluate candidate recall maps on a separate recall-label split. Candidate recall-map sampling is detailed in Appendix B.3. Each candidate is scored by its action prediction error:

$$E_t(\delta) = \mathcal{D}(\pi_\theta(o_t; \delta), a_t), \qquad \delta_t^\star = \arg\min_{\delta \in \mathcal{C}_t} E_t(\delta). \tag{4}$$

Here $\mathcal{C}_t$ is the candidate set, o_t is the current observation, and $\mathcal{D}$ measures the adapted policy's action error against the demonstration action a_t. We retain a nonzero map only if it improves over the no-memory candidate by a margin ϵ; otherwise, the target is zero. This grounds recall supervision in the benefit of history for action prediction.

Stage 3: Teacher-forced memory joint training. We jointly optimize recall prediction and the memory-conditioned policy:

$$\mathcal{L}_{\text{stage3}} = \mathcal{L}_{\text{action}} + \lambda_\delta \sum_{p=1}^{P} w_t[p] \,\text{Huber}\Big(\phi(\hat{\delta}_t[p]), \phi(\delta_t^\star[p])\Big). \tag{5}$$

Here $w_t[p]$ weights pseudo-label confidence, ϕ log-scales recall times, and λ_δ balances the losses. Teacher forcing supplies pseudo-label recall maps while the predictor is being trained. Scheduled

sampling gradually introduces predicted maps to reduce the training–inference mismatch. At inference, STAM uses its predicted recall map.

4 EXPERIMENTS

The goal of our experimental evaluations is to assess whether STAM improves memory-dependent manipulation while preserving general manipulation ability. We assess its performance across simulation benchmarks and real-world robot tasks in Sec. 4.1. To examine the benefits of selective recall, we compare STAM with policies using only the current observation or alternative history-access methods in Sec. 4.2. We then visualize how the spatio-temporal recall map accesses historical visual evidence in Sec. 4.3. Finally, we ablate the memory representations, recall design, and training search budget in Sec. 4.4.

4.1 HOW STAM PERFORMS

Simulation experiments. We evaluate general manipulation and memory-dependent control across four simulation benchmarks. LIBERO (Liu et al., 2023) tests general manipulation across four suites: Spatial, Object, Goal, and Long. LIBERO-MEM (Chung et al., 2025) contains ten tasks testing memory for object motion, action sequences, object relations, and occlusion. Our MIKASA-Robo (Cherepanov et al., 2025) evaluation covers five tasks testing shell-game tracking, motion inference, and color memory. We also evaluate five MemMimic (Gao et al., 2026) tasks involving target matching, delayed matching, place-back, iterative pushing, and iterative flinging. Delayed matching requires retaining an earlier cue, while the iterative tasks require using outcomes from previous attempts. LIBERO-MEM reports average subgoal completion; the other benchmarks report task success rates. Here, * marks our visual-memory-only Pi-Mem reproduction. Evaluation protocols are detailed in Appendix C.1.

Table 1: **Results on LIBERO and LIBERO-MEM.** LIBERO reports average success rate across four suites, while LIBERO-MEM reports average subgoal completion. All values are percentages.

Model	LIBERO Avg.	LIBERO-MEM Avg.
GR00T N1.5 (NVIDIA GEAR, 2025)	95.7	-
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	96.9	-
CronusVLA (Li et al., 2025)	97.0	-
MemoryVLA (Shi et al., 2026)	96.7	-
HAMLET (Koo et al., 2026)	97.6	-
ContextVLA (Jang et al., 2025)	97.0	-
π_0 (Black et al., 2024)	94.2	5.0
SlotVLA (Chung et al., 2025)	-	5.0
Naive E-SlotSSM (Chung et al., 2025)	79.4	14.8
Pi-Mem($\pi_{0.5}$)* (Torne et al., 2026)	93.6	17.3
Ours(GR00T N1.5 (NVIDIA GEAR, 2025))	96.6	28.7
Ours($\pi_{0.5}$ (Physical Intelligence et al., 2025))	**98.2**	**32.4**

STAM maintains strong general manipulation performance and improves memory-dependent control across both backbones. With $\pi_{0.5}$, it achieves 98.2% success on LIBERO. On LIBERO-MEM, it reaches 32.4% subgoal completion, compared with Pi-Mem's 17.3%, as shown in Table 1. On MIKASA-Robo, average success increases by 33.2 percentage points for $\pi_{0.5}$ and 28.0 points for GR00T N1.5 relative to their original policies, as shown in Table 2.

Table 2: **MIKASA-Robo simulation results.** Success rates are reported in percentage. Average is the arithmetic mean over all five MIKASA-Robo tasks.

Model	ShellGame Touch	Intercept Medium	Remember Color3	Remember Color5	Remember Color9	Average
Octo-small (Ghosh et al., 2024)	46.0	39.0	45.0	17.0	11.0	31.6
OpenVLA (Kim et al., 2024)	47.0	14.0	59.0	16.0	6.0	28.4
CronusVLA (Li et al., 2025)	32.0	5.0	31.0	13.0	9.0	18.0
SpatialVLA (Qu et al., 2025)	23.0	27.0	27.0	17.0	11.0	21.0
π_0 (Black et al., 2024)	33.0	42.0	35.0	22.0	15.0	29.4
MemoryVLA (Shi et al., 2026)	88.0	24.0	44.0	30.0	20.0	41.2
GMP (Gao et al., 2026)	**98.0**	83.0	80.0	61.0	17.0	67.8
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	45.0	51.0	47.0	43.0	12.0	39.6
GR00T N1.5 (NVIDIA GEAR, 2025)	48.0	43.0	52.0	39.0	11.0	38.6
Pi-Mem($\pi_{0.5}$)* (Torne et al., 2026)	52.0	60.0	61.0	51.0	12.0	47.2
Ours($\pi_{0.5}$ (Physical Intelligence et al., 2025))	96.0	**88.0**	**87.0**	**73.0**	**20.0**	**72.8**
Ours(GR00T N1.5 (NVIDIA GEAR, 2025))	94.0	85.0	79.0	57.0	18.0	66.6

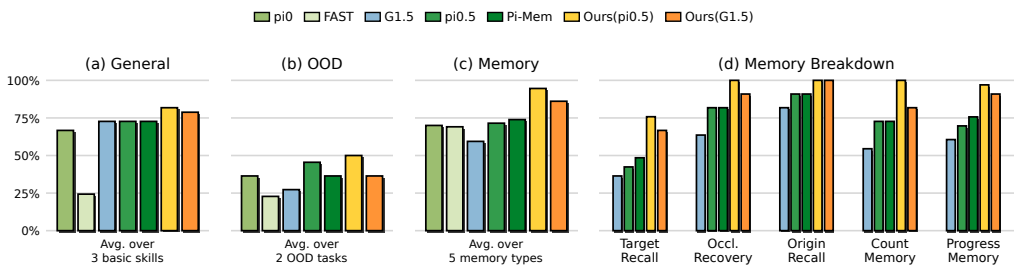


Figure 3: Real-world evaluation by capability group. Panels (a)–(c) show group averages; panel (d) breaks down memory-specific task categories.

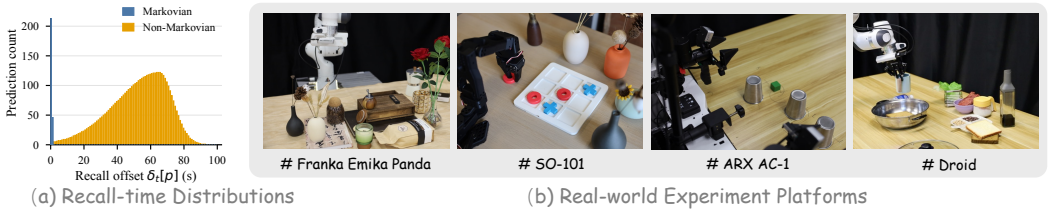
STAM also improves over both original backbones on MemMimic, as shown in Table 3. The gains on delayed matching and iterative manipulation support its ability to use earlier cues and interaction outcomes. We distinguish GMP's reported single-task results from our multi-task reproduction. In the multi-task setting, STAM outperforms our GMP reproduction by 39.7 and 37.4 percentage points with $\pi_{0.5}$ and GR00T N1.5, respectively.

Table 3: **MemMimic simulation results.** Episode-level success rates are reported in percentage. Average is computed over all reported task columns; missing entries are excluded.

Model	T1 Match Color	T1' Random Delay	T2 Place Back	T4 Iter. Pushing	T5 Iter. Flinging	Average
No-hist DP (Chi et al., 2025)	15.3	–	17.3	2.0	61.3	24.0
Mid-hist DP (Chi et al., 2025)	36.3	–	61.7	3.7	56.0	39.4
Mid-hist PTP (Torne et al., 2025)	31.7	–	58.7	1.3	60.7	38.1
Long-hist DP (Chi et al., 2025)	100.0	–	100.0	29.7	66.0	73.9
Long-hist PTP (Torne et al., 2025)	100.0	–	100.0	56.7	71.3	82.0
BC-RNN (Mandlekar et al., 2021)	97.0	–	99.7	48.7	75.3	80.2
GMP (single-task) (Gao et al., 2026)	100.0	**99.0**	98.7	**93.7**	77.3	93.7
GMP (multi-task) (Gao et al., 2026)	62.3	60.9	61.7	55.1	32.4	54.5
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	100.0	78.0	99.7	61.3	71.3	82.1
GR00T N1.5 (NVIDIA GEAR, 2025)	98.7	69.0	100.0	64.7	69.0	80.3
Pi-Mem($\pi_{0.5}$)* (Torne et al., 2026)	100.0	61.0	100.0	72.0	73.0	81.2
Ours($\pi_{0.5}$ (Physical Intelligence et al., 2025))	100.0	98.0	100.0	89.7	**83.5**	**94.2**
Ours(GR00T N1.5 (NVIDIA GEAR, 2025))	100.0	91.0	100.0	91.3	77.3	91.9

Real-world robot experiments. As shown in Fig. 4(b), we conduct real-world experiments on Franka Emika Panda, SO-101, DROID, and ARX AC-1. The tasks are organized into three groups: General Manipulation, OOD Robustness, and Memory Manipulation. Each task is evaluated over 11 trials using final task success. Figure 3 summarizes the results for these groups.

General Manipulation includes three tasks for which the current observation is usually sufficient. They cover scene-based manipulation, instruction grounding, and actuation precision, testing whether memory augmentation preserves basic manipulation skills. **OOD Robustness** includes two tasks, Flip Bottle and Make Tea, intended to test sensitivity to unreliable or weakly relevant history. **Memory Manipulation** includes nine tasks whose correct actions depend on historical information that is not fully available in the current observation. They cover five memory capabilities: Target Recall and Progress Memory each include three tasks, while Occlusion Recovery, Origin Recall, and Count Memory each include one. Task definitions and scoring details are provided in Appendix C.2. The grouped results follow the simulation trend: STAM is most useful on memory manipulation tasks while remaining competitive on general manipulation and OOD robustness.



(a) Recall-time Distributions (b) Real-world Experiment Platforms

Figure 4: **Recall-time distributions and real-world robot platforms. (a)** Schematic recall-offset distributions for Markovian and non-Markovian tasks. Zero offset disables memory access, while positive offsets specify look-back durations. The counts are illustrative rather than measured activation statistics. **(b)** Real-world experiments on Franka Panda, SO-101, ARX AC-1, and DROID.

4.2 Does STAM Benefit from Selective Recall?

STAM improves memory-dependent manipulation without sacrificing general manipulation performance. Simply adding history does not consistently provide the same benefits. Two real-world tasks illustrate this contrast using the $\pi_{0.5}$ backbone. On Add Ingredients, which requires remembering completed interactions, STAM achieves 100.0% success, compared with 72.7% for both the original policy and Pi-Mem and 63.6% for Fixed-Window Recall. On Flip Bottle, STAM preserves the original policy's 54.5% success rate, while Pi-Mem and Fixed-Window Recall achieve 45.5% and 9.1%, respectively. These comparisons in Tables S5 and S8 of the appendix show that STAM benefits from history on a memory-dependent task and avoids the degradation seen with other history-access methods on Flip Bottle.

Figure 4(a) schematically illustrates the intended memory-use pattern behind this design. The blue distribution concentrates near zero to depict reduced memory access on Markovian tasks. The orange distribution extends to larger offsets to depict recall of earlier observations on non-Markovian tasks. In STAM's spatio-temporal recall map, a zero offset disables memory for a patch, while a positive offset specifies how far back to retrieve visual evidence.

4.3 Visualization of Spatio-Temporal Recall Map

We examine how STAM selectively accesses historical visual evidence across space and time. Figure 5 visualizes its recall behavior in a shell-game task. The green block is initially visible before being covered by a cup. After three cup swaps, the robot must identify the target cup from history because the current observation alone no longer reveals the block's location.

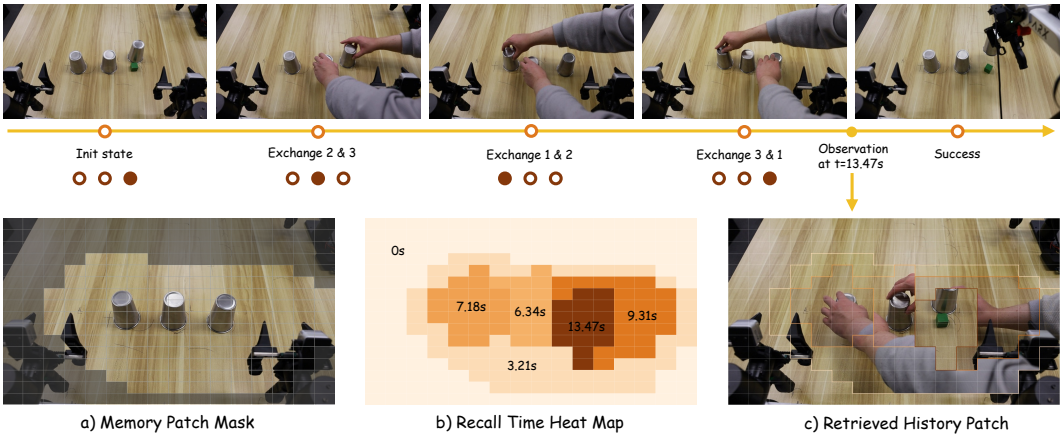


Figure 5: **Visualization of the spatio-temporal recall map. Top:** shell-game sequence; filled dots mark the cup hiding the green block. **Bottom:** memory access at $\tau_t = 13.47\,\mathrm{s}$. **(a)** Active patches access memory; gray patches have $\delta_t[p] = 0$. **(b)** Patch-level look-back durations. **(c)** Retrieved patches shown as an image mosaic for visualization; the policy fuses their token features.

Spatially, the recall map activates memory around the cups while suppressing background access. Temporally, different active regions retrieve visual evidence from the initial observation and intermediate cup swaps. Together, these recalled observations help the policy identify which cup hides the green block.

4.4 Ablation Study

We evaluate the contributions of coarse-fine visual memory, spatio-temporal recall, and the Stage-2 search budget. Figure 6 summarizes the results on MIKASA-Robo, two real-world tasks, and MemMimic. Each component ablation changes one memory mechanism in the complete STAM policy, while the search-budget ablation keeps the architecture fixed. Appendix D.3 provides detailed task-level results and additional ablations of the recall loss, confidence weighting, teacher forcing, and SSMC architecture.

Memory representation. We separately remove state-compressed coarse memory from deeper vision layers and retrieved fine patch memory from shallow layers. Both variants retain the spatio-

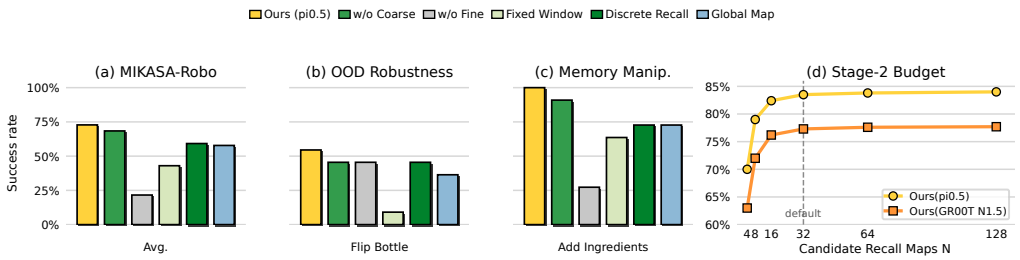


Figure 6: **Ablations of memory representation, recall design, and Stage-2 search budget. (a)** Average success on five MIKASA-Robo tasks. **(b)** Success on Flip Bottle. **(c)** Success on Add Ingredients. Panels (a)–(c) compare complete STAM with variants that remove a memory source or modify recall, using the $\pi_{0.5}$ backbone. **(d)** Success on MemMimic T5 Iterative Flinging as the number of candidate recall maps increases, with diminishing gains beyond the budget of $N = 32$.

temporal recall map and the remaining memory path. On MIKASA-Robo, removing coarse memory reduces average success from 72.8% to 68.4%, while removing fine memory reduces it to 21.6%. Both memory paths contribute to performance, with a larger drop when fine memory is removed. These results indicate that compressed historical context alone cannot replace directly retrievable fine visual evidence on these tasks.

Spatio-temporal recall. Fixed-Window Recall and Discrete Recall modify temporal retrieval, while Global Recall Map removes patch-level spatial selection. Discrete Recall predicts a frame index instead of continuous physical time, and Global Recall Map shares one recall decision across the image. All three variants perform worse than complete STAM on MIKASA-Robo and both real-world tasks. On MIKASA-Robo, Fixed-Window Recall and Global Recall Map achieve 43.0% and 57.8% average success, respectively, compared with 72.8% for STAM. These comparisons support both adaptive temporal retrieval and patch-level spatial selection.

Training search budget. We vary the number of candidate recall maps N used by Action-Guided Recall Label Search on MemMimic T5 Iterative Flinging. Performance improves as the budget increases, with diminishing gains beyond $N = 32$. Increasing N from 32 to 128 improves success by only 0.5 percentage points for $\pi_{0.5}$ and 0.4 for GR00T N1.5. We therefore use $N = 32$ to obtain performance close to that of larger budgets with fewer candidate evaluations.

5 Conclusion and Limitation

We presented STAM, a plug-in memory module for generalist robot policies. Its coarse-fine visual memory combines compressed historical context with directly retrievable visual details. A spatio-temporal recall map selects which image patches access memory and how far back to retrieve fine visual evidence. Layerwise coarse-fine memory fusion augments the vision encoder without modifying the policy backbone. An action-guided training procedure learns recall decisions from action errors. Experiments in simulation and real-world demonstrate improved memory-dependent manipulation across different backbones while maintaining strong general manipulation performance.

A practical limitation of STAM is the offline computation required for recall-label search. Stage 2 evaluates candidate recall maps with a frozen policy and requires no backpropagation. This search is confined to training; at inference, the recall map is predicted directly. Future work could reduce this cost through more efficient candidate selection and recall supervision.

AI use statement

Generative AI tools were used to assist with manuscript translation, language and structural editing, literature search, and LaTeX formatting. They also assisted with the discussion and interpretation of experimental results. The authors reviewed and verified the AI-assisted content and take responsibility for the final manuscript and reported results.

REFERENCES

Ibrahim Alabdulmohsin, Xiaohua Zhai, Alexander Kolesnikov, and Lucas Beyer. Getting ViT in shape: Scaling laws for compute-optimal model design. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.13035.

Shir Amir, Yossi Gandelsman, Shai Bagon, and Tali Dekel. Deep ViT features as dense visual descriptors, 2021. URL https://arxiv.org/abs/2112.05814.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2024. URL https://arxiv.org/abs/2410.24164.

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, 2022. URL https://arxiv.org/abs/2212.06817.

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, 2023. URL https://arxiv.org/abs/2307.15818.

Egor Cherepanov, Nikita Kachaev, Alexey K. Kovalev, and Aleksandr I. Panov. Memory, benchmark & robots: A benchmark for solving complex tasks with reinforcement learning, 2025. URL https://arxiv.org/abs/2502.10550.

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), 2025. doi: 10.1177/02783649241273668. URL https://journals.sagepub.com/doi/10.1177/02783649241273668.

Nhat Chung, Taisei Hanyu, Toan Nguyen, Huy Le, Frederick Bumgarner, Duy Minh Ho Nguyen, Khoa Vo, Kashu Yamazaki, Chase Rainwater, Tung Kieu, Anh Nguyen, and Ngan Le. Rethinking progression of memory state in robotic manipulation: An object-centric perspective, 2025. URL https://arxiv.org/abs/2511.11478.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2405.21060.

Daniel Y. Fu, Tri Dao, Khaled K. Saab, Armin W. Thomas, Atri Rudra, and Christopher Ré. Hungry hungry hippos: Towards language modeling with state space models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2212.14052.

Yihuai Gao, Jinyun Liu, Shuang Li, and Shuran Song. Gated memory policy, 2026. URL https://arxiv.org/abs/2604.18933.

Dibya Ghosh, Homer Rich Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Quan Vuong, Ted Xiao, Pannag R. Sanketi, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, 2024. doi: 10.15607/RSS.2024.XX.090. URL https://www.roboticsproceedings.org/rss20/p090.html.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2023. URL https://arxiv.org/abs/2312.00752.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2111.00396.

Huiwon Jang, Sihyun Yu, Heeseung Kwon, Hojin Jeon, Younggyo Seo, and Jinwoo Shin. Contextvla: Vision-language-action model with amortized multi-frame context, 2025. URL https://arxiv.org/abs/2510.04246.

Jindong Jiang, Fei Deng, Gautam Singh, Minseung Lee, and Sungjin Ahn. Slot state space models. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2406.12272.

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 *Conference on Robot Learning*, 2024. URL https://openreview.net/forum?id=ZMnD6QZAE6.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success, 2025. URL https://arxiv.org/abs/2502.19645.

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*, 2026. URL https://openreview.net/forum?id=KcJ9U0x6kO.

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, 2026. URL https://arxiv.org/abs/2603.12942.

Hao Li, Shuai Yang, Yilun Chen, Xinyi Chen, Xiaoda Yang, Yang Tian, Hanqing Wang, Tai Wang, Dahua Lin, Feng Zhao, and Jiangmiao Pang. Cronusvla: Towards efficient and robust manipulation via multi-frame vision-language-action modeling, 2025. URL https://arxiv.org/abs/2506.19816.

Shufan Li, Harkanwar Singh, and Aditya Grover. Mamba-ND: Selective state space modeling for multi-dimensional data, 2024. URL https://arxiv.org/abs/2402.05892.

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

Yue Liu, Yunjie Tian, Yuzhong Zhao, Hongtian Yu, Lingxi Xie, Yaowei Wang, Qixiang Ye, Jianbin Jiao, and Yunfan Liu. VMamba: Visual state space model. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2401.10166.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martin-Martin. What matters in learning from offline human demonstrations for robot manipulation. In *Conference on Robot Learning*, 2021. URL https://openreview.net/forum?id=JrsfBJtDFdI.

NVIDIA GEAR. Gr00t n1.5: An improved open foundation model for generalist humanoid robots, 2025. URL https://research.nvidia.com/labs/gear/gr00t-n1_5/.

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, 2025. URL https://arxiv.org/abs/2501.09747.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael 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, 2025. URL https://arxiv.org/abs/2504.16054.

Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Yan Ding, Zhigang Wang, Jiayuan Gu, Bin Zhao, and Dong Wang. Spatialvla: Exploring spatial representations for visual-language-action model, 2025. URL https://arxiv.org/abs/2501.15830.

Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks?, 2021. URL https://arxiv.org/abs/2108.08810.

Hao Shi, Bin Xie, Yingfei Liu, Lin Sun, Fengrong Liu, Tiancai Wang, Erjin Zhou, Haoqiang Fan, Xiangyu Zhang, and Gao Huang. Memoryvla: Perceptual-cognitive memory in vision-language-action models for robotic manipulation. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=54U3XHf7qq.

Jimmy T. H. Smith, Andrew Warrington, and Scott W. Linderman. Simplified state space layers for sequence modeling. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2208.04933.

Wei Song, Yuran Wang, Zijia Song, Yadong Li, Zenan Zhou, Long Chen, Jianhua Xu, Jiaqi Wang, and Kaicheng Yu. DualToken: Towards unifying visual understanding and generation with dual visual vocabularies, 2025. URL https://arxiv.org/abs/2503.14324.

Marcel Torne, Andy Tang, Yuejiang Liu, and Chelsea Finn. Learning long-context diffusion policies via past-token prediction, 2025. URL https://arxiv.org/abs/2505.09561.

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

Michael Tschannen, Alexey Gritsenko, Xiao Wang, Muhammad Ferjad Naeem, Ibrahim Alabdulmohsin, Nikhil Parthasarathy, Talfan Evans, Lucas Beyer, Ye Xia, Basil Mustafa, Olivier Hénaff, Jeremiah Harmsen, Andreas Steiner, and Xiaohua Zhai. SigLIP 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features, 2025. URL https://arxiv.org/abs/2502.14786.

Jue Wang, Wentao Zhu, Pichao Wang, Xiang Yu, Linda Liu, Mohamed Omar, and Raffay Hamid. Selective structured state-spaces for long-form video understanding. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023. URL https://arxiv.org/abs/2303.14526.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023. URL https://arxiv.org/abs/2303.15343.

Lianghui Zhu, Bencheng Liao, Qian Zhang, Xinlong Wang, Wenyu Liu, and Xinggang Wang. Vision Mamba: Efficient visual representation learning with bidirectional state space model. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2401.09417.

APPENDIX

This page links the appendix sections and their subsections for quick navigation.

- **Appendix A. Method Details and Visualizations**
  A.1 Recall-Map Visualization Details; A.2 Layerwise Coarse-Fine Memory Fusion; A.3 Causal Memory Access and Inference Order; A.4 Episodic Patch Buffer and Fine Retrieval; A.5 Complexity and Latency.
- **Appendix B. Training Details**
  B.1 Dataset Splits and Stage Schedule; B.2 Random-Recall Memory Adaptation; B.3 Action-Guided Recall Label Search; B.4 Teacher-Forced Memory Joint Training.
- **Appendix C. Evaluation Protocols**
  C.1 Simulation Benchmarks; C.2 Real-World Robot Evaluation Protocol.
- **Appendix D. Additional Results and Ablations**
  D.1 Full Simulation Results; D.2 Full Real-World Task Results; D.3 Full Ablation Task Results.

A METHOD DETAILS AND VISUALIZATIONS

This appendix supplements the architecture in Fig. 2 of the main paper with visualization procedures, layerwise fusion details, causal inference pseudocode, buffer implementation, and computational cost.

A.1 RECALL-MAP VISUALIZATION DETAILS

The shell-game visualization in the main paper illustrates how the recall map accesses historical visual evidence. The policy operates on 224×224 inputs with 14×14 ViT patches, producing a 16×16 grid of 256 patches. For visualization, we remap the recall map to the original 640×480 camera image. Panel (c) assembles historical image patches corresponding to the retrieved fine patch tokens, using the same spatial indices and the historical frames selected from the episodic patch buffer.

A.2 LAYERWISE COARSE-FINE MEMORY FUSION

Layerwise coarse-fine fusion decides not only which memory source is used, but also where it enters the vision encoder. Let $z_{t,\ell}[p]$ be the current token of patch p after ViT layer ℓ, $f_t[p]$ be the retrieved fine patch memory from the episodic patch buffer, and $c_t[p]$ be the state-compressed coarse memory from the SSMC. Only patches in the active set $\mathcal{A}_t$ receive memory. At a selected fusion layer, STAM applies a residual cross-attention adapter:

$$z_{t,\ell}[p] \leftarrow z_{t,\ell}[p] + \gamma_\ell \mathbf{1}[p \in \mathcal{A}_t]\, \mathrm{CrossAttn}_\ell\left(\mathrm{LN}(z_{t,\ell}[p]),\, \mathrm{LN}(\mathcal{M}_{t,\ell}[p])\right),$$
$$\mathcal{M}_{t,\ell}[p] = \begin{cases} f_t[p], & \ell \in \mathcal{L}_{\text{fine}}, \\ c_t[p], & \ell \in \mathcal{L}_{\text{coarse}}. \end{cases} \tag{S1}$$

Here γ_ℓ is a learnable scalar initialized at zero, so the initial model is equivalent to the original VLA vision encoder before the memory adapters are trained. The memory in Eq. S1 is causal: fine memory is retrieved from $\mathcal{B}_{t-1}$, and coarse memory is read from SSMC state before x_t is written.

For both $\pi_{0.5}$ and GR00T N1.5, the vision tower follows the SigLIP/SoViT-style 224×224, patch-14, 27-layer ViT geometry used by the underlying VLA backbones (Zhai et al., 2023; Alabdulmohsin et al., 2023; Tschannen et al., 2025; Physical Intelligence et al., 2025; NVIDIA GEAR, 2025). Let $N = 27$ be the number of ViT blocks and let K denote the boundary between the lower-to-middle and deep parts of the encoder. We use this shared geometry to set a fixed layer split. Following analyses of ViT and SigLIP representations, lower and middle layers retain spatially localized perceptual information, while later layers are more aligned with semantic content (Raghu et al., 2021; Amir et al., 2021; Song et al., 2025). In particular, recent SigLIP analysis partitions a 27-layer SigLIP encoder into shallow layers 1–7, middle layers 8–17, and deep layers 18–27;

it further shows that features around layer 6 cluster by low-level cues such as color and texture, whereas features around layer 26 cluster by semantic content (Song et al., 2025). We therefore set $K = 17$. By default, STAM cross-attends to retrieved fine patch memory every two layers in the first K layers, and cross-attends to state-compressed coarse memory every four layers in the remaining $N - K$ layers. Concretely, retrieved fine patch memory is injected at layers 2, 4, 6, 8, 10, 12, 14, and 16, while state-compressed coarse memory is injected at layers 18, 22, and 26, i.e., $\mathcal{L}_{\text{fine}} = \{2, 4, 6, 8, 10, 12, 14, 16\}$ and $\mathcal{L}_{\text{coarse}} = \{18, 22, 26\}$.

This assignment matches the role of the two memory sources. Retrieved fine patch memory preserves local historical evidence at the same flattened view-patch index, so injecting it before the deep semantic layers lets the current visual stream recover details such as occluded appearance, contact traces, or precise historical object position. State-compressed coarse memory summarizes longer-horizon visual history through the SSMC state, so injecting it in the deep layers gives high-level tokens access to task progress and historical context without forcing all low-level features to attend to a compressed state. The same active patch set controls both paths, which keeps memory access sparse and prevents inactive regions from being modified by fine retrieval or coarse fusion.

We further evaluate whether this layer allocation matters in Table S1. All variants use STAM on $\pi_{0.5}$ and are evaluated only on MIKASA-Robo, so the table isolates the allocation of fine and coarse memory inside the same policy backbone and benchmark.

Table S1: **Extended ablation of layerwise coarse-fine memory fusion.** All layer fusion variants are based on STAM($\pi_{0.5}$) and are evaluated on MIKASA-Robo.

Setting	Fine-memory layers	Coarse-memory layers	ShellGame Touch	Intercept Medium	Remember Color3	Remember Color5	Remember Color9	Average
Default Layerwise Fusion	2–16, stride=2	18–26, stride=4	**96.0**	**88.0**	**87.0**	**73.0**	**20.0**	**72.8**
Joint Fine-Coarse Fusion	2–16, stride=2;	18–26, stride=4	79.0	70.0	68.0	51.0	14.0	56.4
Uniform-4 Fusion	4–16, stride=4	18–26, stride=4	90.0	81.0	79.0	62.0	17.0	65.8
Uniform-2 Fusion	2–16, stride=2	18–26, stride=2	94.0	86.0	85.0	70.0	19.0	70.8
Dense-Fine Fusion	1–17, stride=1	18–26, stride=4	93.0	83.0	82.0	66.0	18.0	68.4
Dense-Coarse Fusion	2–16, stride=2	18–27, stride=1	92.0	84.0	83.0	66.0	18.0	68.6
Sparse-Coarse Fusion	2–16, stride=2	18–26, stride=8	85.0	76.0	74.0	56.0	16.0	61.4

Ablation settings. *Default Layerwise Fusion* is the configuration described above: retrieved fine patch memory is fused at layers $\{2, 4, 6, 8, 10, 12, 14, 16\}$, and state-compressed coarse memory is fused at layers $\{18, 22, 26\}$. *Joint Fine-Coarse Fusion* removes the lower-to-middle/deep separation while keeping the same selected layer indices; both retrieved fine patch memory and state-compressed coarse memory are fused at layers $\{2, 4, 6, 8, 10, 12, 14, 16, 18, 22, 26\}$. *Uniform-4 Fusion* follows the four-layer fusion interval used by Pi-Mem-style temporal attention designs (Torne et al., 2026): fine memory is fused at layers $\{4, 8, 12, 16\}$, and coarse memory remains at $\{18, 22, 26\}$. *Uniform-2 Fusion* keeps the default fine-memory layers $\{2, 4, 6, 8, 10, 12, 14, 16\}$ but also applies coarse memory every two layers in the deep part, $\{18, 20, 22, 24, 26\}$. *Dense-Fine Fusion* fuses retrieved fine patch memory at every lower-to-middle layer, $\{1, \ldots, 17\}$, while keeping coarse memory at $\{18, 22, 26\}$. *Dense-Coarse Fusion* keeps fine memory at $\{2, 4, 6, 8, 10, 12, 14, 16\}$ and fuses state-compressed coarse memory at every deep layer, $\{18, \ldots, 27\}$. *Sparse-Coarse Fusion* keeps the default fine-memory layers and makes coarse memory sparser, using only layers $\{18, 26\}$.

Results. The default split obtains the highest average score, 72.8, indicating that the proposed allocation is not only a notation choice but an empirically useful way to expose different memory granularity to the encoder. Removing the separation between fine and coarse memory gives the lowest average score, 56.4, a drop of 16.4 percentage points from the default. This suggests that simply making both memory sources available at all selected fusion layers can blur their roles: low-level tokens receive compressed state information too early, while deep tokens are also forced to process local patch evidence that is better handled before semantic abstraction. Among the interval ablations, *Uniform-2 Fusion* is closest to the default at 70.8, but it still trails by 2.0 points, suggesting that denser deep-layer coarse fusion can be tolerated but does not improve the default schedule. *Dense-Coarse Fusion* and *Dense-Fine Fusion* reach similar averages, 68.6 and 68.4, showing that injecting either memory source at every available layer is less useful than matching memory granu-

larity to encoder depth. *Uniform-4 Fusion* drops to 65.8, consistent with the need for denser lower-to-middle fine-memory access when local historical evidence matters. *Sparse-Coarse Fusion* drops further to 61.4, showing that overly sparse deep-layer coarse memory weakens the long-horizon context used for task progress and memory-heavy color tracking. Together, these results support the default coarse-fine design: fine memory is most useful before deep semantic abstraction, while coarse memory should enter later and less frequently.

A.3 Causal Memory Access and Inference Order

STAM uses memory only through states that were available before the current action is decoded. This constraint is important because both memory paths are powerful enough to leak current-frame information if the read and write order is not fixed. At decision step t, the policy observes I_t and encodes it into current view-patch tokens x_t, but the memory used by the current decision is read from the previous SSMC state h_{t-1} and the previous episodic patch buffer $\mathcal{B}_{t-1}$. The current tokens are written only after the action has been predicted and executed.

The online computation follows a read–decode–write factorization. First, STAM reads $y_t = \mathrm{Read}_\psi(h_{t-1})$ and maps it to $c_t = W_c\mathrm{LN}(y_t)$. It then predicts $\hat{\delta}_t$ and forms the active patch set $\mathcal{A}_t$ and effective recall map δ_t as in Eq. 2 of the main paper, with memory access disabled when the buffer is empty. For each active patch, it retrieves $f_t[p] = \mathrm{Retrieve}(\mathcal{B}_{t-1}, \tau_t - \delta_t[p], p)$ from the previous buffer. The only remaining computation that changes the memory state is:

$$\begin{aligned} z_t &= \mathrm{LayerwiseFuse}_\theta(x_t, \{f_t[p]\}_{p\in\mathcal{A}_t}, c_t, \mathcal{A}_t), \qquad \hat{a}_t = \pi_\theta(z_t, q), \\ \mathcal{B}_t &= \mathcal{B}_{t-1} \cup \{(\tau_t, x_t)\}, \qquad h_t = \mathrm{Update}_\psi(h_{t-1}, x_t). \end{aligned} \tag{S2}$$

The first line of Eq. S2 decodes the action from the current tokens and already-read memory. The second line is a post-decision write: it makes the current observation available to future decisions, but not to the current one.

Algorithm S1 gives the full pseudocode for this order. The coarse path is causal because c_t is a projection of $y_t = \mathrm{Read}_\psi(h_{t-1})$, not of the updated state h_t. The fine path is causal because every retrieval query uses $\mathcal{B}_{t-1}$ and a target time $\tau_t - \delta_t[p]$. For inactive patches, $\delta_t[p] = 0$ and no retrieval is issued. For active patches, nearest-older lookup follows Eq. 3 of the main paper and only selects entries already in $\mathcal{B}_{t-1}$, so the retrieved frame satisfies $\tau_i < \tau_t$. Thus the recall map can decide which historical evidence to use, but it cannot read the current frame after it has been written to memory.

Algorithm S1 STAM inference with causal visual memory.

Require: trained VLA policy π_θ, vision encoder V_θ, state-space memory core, episodic patch buffer $\mathcal{B}$, optional instruction q

Initialize h_0 and $\mathcal{B}_0 \leftarrow \emptyset$
for decision step $t = 1, \ldots, T$ **do**
  Observe multi-view image set I_t and timestamp τ_t
  $x_t \leftarrow V_\theta(I_t)$ ▷ view-patch tokens, $x_t \in \mathbb{R}^{P\times d}$
  $y_t \leftarrow \mathrm{Read}_\psi(h_{t-1})$ ▷ causal readout from past state
  $c_t \leftarrow W_c\mathrm{LN}(y_t)$ ▷ State-Compressed Coarse Memory
  $\hat{\delta}_t \leftarrow (\tau_t - \tau_0)\mathrm{HardSigmoid}(W_\delta\mathrm{LN}(y_t))$
  $\delta_t[p] \leftarrow 0$ for all p with $\hat{\delta}_t[p] \le \epsilon_\delta$; otherwise $\delta_t[p] \leftarrow \hat{\delta}_t[p]$
  $\mathcal{A}_t \leftarrow \{p : \delta_t[p] > \epsilon_\delta\}$
  if $\mathcal{B}_{t-1} = \emptyset$ **then** $\mathcal{A}_t \leftarrow \emptyset$ and $\delta_t \leftarrow 0$
  for each active view-patch $p \in \mathcal{A}_t$ **do**
    $f_t[p] \leftarrow \mathrm{Retrieve}(\mathcal{B}_{t-1},\ \tau_t - \delta_t[p],\ p)$ ▷ Eq. 3 of the main paper
  end for
  $z_t \leftarrow \mathrm{LayerwiseFuse}_\theta(x_t, \{f_t[p]\}_{p\in\mathcal{A}_t}, c_t, \mathcal{A}_t)$
  $\hat{a}_t \leftarrow \pi_\theta(z_t, q)$ ▷ diffusion/flow or autoregressive decoding
  Execute $\hat{a}_t$
  $\mathcal{B}_t \leftarrow \mathcal{B}_{t-1} \cup \{(\tau_t, x_t)\}$
  $h_t \leftarrow \mathrm{Update}_\psi(h_{t-1}, x_t)$
end for

A.4 Episodic Patch Buffer and Fine Retrieval

The episodic patch buffer preserves local visual evidence that should not be compressed into the recurrent SSMC state. Each buffer entry stores a timestamp and the corresponding view-patch tokens, written compactly as $\mathcal{B}_{t-1} = \{(\tau_i, b_i)\}_{i<t}$ with $b_i = \{b_i[p]\}_{p=1}^{P}$, where p is the flattened view-patch index over all camera views and image patches. In the notation of the main paper and Algorithm S1, b_i is written as x_i for simplicity. Implementation-wise, the stored token is the shallow view-patch feature used by the fine-memory adapters: it is after patch embedding and early visual projection, but before the deep semantic layers where state-compressed coarse memory is injected. This choice keeps local appearance, position, and contact evidence available for later retrieval while avoiding storage of raw pixels or final high-level VLA tokens.

Fine retrieval converts the continuous physical recall time predicted by the recall map into an index in the discrete buffer. For an active patch $p \in \mathcal{A}_t$, the target timestamp is simply $s_t[p] = \tau_t - \delta_t[p]$. Because the buffer contains only sampled decision frames, $s_t[p]$ usually falls between two stored timestamps. For a nonempty buffer, STAM applies the nearest-older lookup in Eq. 3 of the main paper. When a frame exists at or before the target timestamp, the selected frame is the closest available frame that is at least as old as the requested recall time. If $s_t[p]$ is earlier than τ_1, clamping the target to τ_1 selects the earliest available buffer entry; if $s_t[p]$ matches a stored timestamp exactly, retrieval is exact. If the buffer is empty, STAM sets $\mathcal{A}_t = \emptyset$ and $\delta_t = 0$, so no lookup is performed.

This retrieval rule is intentionally conservative. Except for the earliest-frame fallback, choosing the nearest older frame prevents the fine path from using a frame that is more recent than the requested historical evidence, which is useful when the recall map points to a state before an occlusion, object swap, or contact event. It also avoids interpolating visual tokens across two different physical states. Inactive patches never call this lookup; for them $f_t[p] = \varnothing$, and the corresponding visual token is processed without fine-memory cross-attention. Thus the buffer may grow with episode length, but the retrieval cost is governed by $|\mathcal{A}_t|$, the number of patches whose current decision asks for fine historical evidence.

A.5 Complexity and Latency

STAM separates memory storage from dense memory attention. The SSMC keeps a fixed-size recurrent state per view-patch stream for coarse history, while the episodic patch buffer stores historical patch tokens linearly with episode length. At inference time, however, only patches in the active set $\mathcal{A}_t$ retrieve fine memory and participate in memory fusion. This makes the fine-memory computation scale with the active ratio, rather than with all patches across all historical frames.

Table S2 reports the measured model-side complexity and latency on a single NVIDIA A800 GPU. We use batch size 1, three 224×224 camera images, bfloat16 precision, and 10 flow-matching steps for one action-chunk inference. Parameters count unique learnable weights, so modules reused across read and write are counted only once. FLOPs are accumulated over the actual inference calls; in particular, the action expert parameters are counted once, while its FLOPs include all ten steps.

The main cost still comes from the original VLA backbone. The image encoders, observation prefix forward pass, and 10-step action expert together account for 296.112 ms without compilation and 61.316 ms with compilation, matching the measured $\pi_{0.5}$ latency on the same A800 setup. STAM increases the total latency to 314.4 ms and 77.9 ms, respectively. The added memory path is therefore 18.3 ms in the uncompiled path and 16.6 ms in the compiled path, while adding only 61.4M unique parameters beyond the $\pi_{0.5}$ backbone. Most of this overhead comes from layerwise coarse-fine memory fusion; the SSMC read, episodic retrieval, and post-decision write are small because they operate on fixed-size state or the active patch set, contributing only about 29.2 GFLOPs in total. This cost profile supports the intended design: STAM introduces causal visual memory without sending dense frame histories through the backbone. The additional cost is modest relative to the full VLA inference path, while the same $\pi_{0.5}$ backbone gains substantially on non-Markovian manipulation tasks, improving the MIKASA-Robo average from 39.6 to 72.8 in Table 2 of the main paper.

Table S2: **Measured complexity and latency of STAM inference.** Timings are measured on an NVIDIA A800 GPU with three camera images and 10 flow-matching steps.

Model part	Parameters	FLOPs	Inference time (A800)	Inference time (A800, compiled)
Image encoders (3 camera images)	414M	0.66 TFLOPs	58.8 ms	11.5 ms
Observation prefix forward pass (VLA prefix + KV-cache prefill)	2.51B	3.97 TFLOPs	126.5 ms	26.6 ms
STAM causal memory read (SSMC read + coarse-memory / recall map heads)	3.0M	3.42 GFLOPs	3.2 ms	2.8 ms
Episodic fine patch retrieval (active patches only)	0	< 0.01 GFLOPs	0.8 ms	0.7 ms
Layerwise coarse-fine memory fusion (selected ViT layers)	58.4M	22.43 GFLOPs	11.6 ms	11.2 ms
x10 action forward pass (flow action expert)	427M	0.39 TFLOPs	110.8 ms	23.2 ms
Post-decision memory write (SSMC update + episodic buffer append)	0 (shared)	3.32 GFLOPs	2.7 ms	1.9 ms
Inference total	**3.41B**	**5.05 TFLOPs**	**314.4 ms**	**77.9 ms**

B Training Details

This appendix provides dataset splits, recall-map sampling rules, search criteria, and training schedules for the training procedure presented in Sec. 3.3 and Algorithm 1 of the main paper.

B.1 Dataset Splits and Stage Schedule

We split each robot demonstration dataset at the episode level into a training split $\mathcal{D}_{\text{train}}$ and a recall-label split $\mathcal{D}_{\text{val}}$. Unless otherwise stated, we use an 80/20 split. The recall-label split is not the final evaluation set; it is a held-out set used only for Stage-2 pseudo-label construction. This separation is useful because Stage 1 learns a memory-compatible policy on $\mathcal{D}_{\text{train}}$, while Stage 2 asks whether different memory choices improve action prediction on episodes that were not used to fit the Stage-1 action pathway.

Stage 1 trains on $\mathcal{D}_{\text{train}}$ with randomly generated recall maps and uses only the original action loss. Stage 2 freezes the Stage-1 model and searches recall map pseudo labels on $\mathcal{D}_{\text{val}}$. For each validation timestep, we run 12 independent search passes with different candidate samples and keep the accepted labeled records, producing a recall-label dataset $\mathcal{D}_{\text{recall}}$ that is about three times larger than $\mathcal{D}_{\text{train}}$ in number of labeled timesteps. Stage 3 jointly trains from both sources: action learning samples from $\mathcal{D}_{\text{train}}$, while recall map regression samples from $\mathcal{D}_{\text{recall}}$.

The modules updated in each stage follow the role of the stage. In Stage 1, the recall map head is frozen, while the SSMC, coarse-memory readout head, layerwise fusion adapters, and selected vision-encoder parameters are trained to accept memory inputs. In Stage 2, all model parameters are frozen and only pseudo labels are generated. In Stage 3, we train the recall map head together with the trainable memory modules and the selected VLA parameters used in Stage 1. This schedule keeps the search target fixed during pseudo-label construction and avoids changing the policy while it is being used to score candidate memories.

B.2 Random-Recall Memory Adaptation

For each training step, we either use an all-zero recall map or sample a random recall map. The all-zero case is kept with probability 0.25, so the policy continues to see ordinary current-frame inference during memory adaptation. Otherwise, each patch is activated by a Bernoulli variable,

$$g_t[p] \sim \text{Bernoulli}(\rho_s), \qquad \delta_t^{\text{rand}}[p] = g_t[p] \cdot \text{LogUniform}(\delta_{\min}, \tau_t - \tau_0).$$

Here s is the Stage-1 training step, ρ_s is the active-patch ratio, and $\delta_{\min} = 0.25\,\text{s}$. If the current episode age $\tau_t - \tau_0$ is smaller than $\delta_{\min}$, the sampled map is set to all zero because there is no

meaningful older frame to retrieve. We linearly increase ρ_s from 0.05 to 0.30 during the first 60% of Stage 1 and keep it fixed afterwards.

This curriculum addresses a failure mode that a reviewer would naturally worry about. If too many random patches use unrelated history at the beginning of training, the simplest policy response is to ignore the memory pathway. The sparse-to-denser schedule instead first teaches the fusion adapters to tolerate a small number of historical tokens and then exposes the model to broader memory access. The all-zero preservation plays a different role: it keeps the current-frame policy calibrated, so memory adaptation does not force history into tasks or frames where the current observation is already sufficient.

B.3 Action-Guided Recall Label Search

We construct the candidate set $\mathcal{C}_t$ used in Eq. 4 of the main paper as follows.

We use five candidate families. *All-zero no-memory* sets $\delta[p] = 0$ for all patches and is always included. *Global single-time recall* assigns one sampled recall time to the full image; it tests whether the action only needs a coarse episode-level timestamp. *Sparse random patch recall* activates a small set of independent patches and is useful when the relevant evidence is local. *Region/block recall* activates connected patch blocks with a shared or smoothly varying recall time; this family matches cases such as the shell-game visualization in Sec. 4.3 of the main paper, where several contiguous cup regions need different historical frames. *Temporal-smooth recall* perturbs the accepted recall map from nearby timesteps, reflecting that memory demand often changes smoothly as the robot observes a slowly evolving scene. With the default budget $N = 32$, we use 1 all-zero candidate, 6 global candidates, 8 sparse patch candidates, 11 region/block candidates, and 6 temporal-smooth candidates. When no previous accepted map is available, the temporal-smooth slots are replaced by region/block candidates.

Each candidate is scored after full action decoding. For flow-based policies such as $\pi_{0.5}$, this means running all flow-matching steps before computing the action error. For autoregressive action-token policies, the decoded action tokens are first mapped back to the continuous action space. We use the normalized action-chunk Huber error,

$$E_t(\delta) = \frac{1}{H} \sum_{h=1}^{H} \mathrm{Huber}_{\beta_a} \left(\mathrm{Norm}(\hat{a}_{t+h}(\delta)) - \mathrm{Norm}(a_{t+h}) \right),$$

where H is the action horizon, $\hat{a}_{t+h}(\delta)$ is decoded using candidate δ, and a_{t+h} is the demonstration action. The best candidate $\delta_t^\star = \arg\min_{\delta \in \mathcal{C}_t} E_t(\delta)$ is accepted only when

$$E_t(0) - E_t(\delta_t^\star) > \max(\epsilon_{\mathrm{abs}}, \epsilon_{\mathrm{rel}} E_t(0)),$$

with $\epsilon_{\mathrm{abs}} = 0.02$ and $\epsilon_{\mathrm{rel}} = 0.05$ in normalized action units. Otherwise, the pseudo label is set to the all-zero recall map.

The confidence weight $w_t[p]$ down-weights noisy pseudo labels. For an accepted nonzero label, we compute a scalar margin confidence

$$\bar{w}_t = \mathrm{clip}\left(\frac{E_t(0) - E_t(\delta_t^\star)}{E_t(0) + 10^{-6}}, 0, 1 \right),$$

and assign it to patches affected by the winning candidate. If repeated search passes select inconsistent regions or recall times, we multiply this confidence by the empirical selection frequency of the winning region. For all-zero labels, the zero target receives full confidence when no candidate passes the margin test. This rule keeps the supervision simple: high-confidence memory labels are those that repeatedly improve action prediction, while ambiguous candidates contribute less to the recall map regression.

Patch-level credit assignment is handled mainly by the candidate design. An action error is a global scalar, so a purely independent random patch search would not reliably identify which patch caused the improvement. We therefore search region/block candidates before relying on sparse patch refinement. The region candidates first identify contiguous parts of the image whose history matters; sparse patch candidates then refine active area. That's why Stage 2 can produce patch-level recall map labels from action-level supervision without requiring manual annotation of memory regions.

B.4 Teacher-Forced Memory Joint Training

Stage 3 trains the final policy with the pseudo labels from Stage 2. At each training step, the recall map head predicts $\hat{\delta}_t$, but the map actually used for retrieval and fusion may be either the pseudo label or the prediction:

$$\delta_t^{\text{train}} = \begin{cases} \delta_t^\star, & \text{teacher forcing,} \\ \hat{\delta}_t, & \text{scheduled sampling.} \end{cases}$$

Teacher forcing lets the action loss train the memory-augmented policy before the recall map head is accurate. This shortens training because action generation and recall map prediction are optimized in the same stage, rather than waiting for a separately trained recall predictor to become reliable.

Scheduled sampling is used to reduce the train-test mismatch. For the first 20% of Stage 3, we use teacher forcing for all samples. The probability of using the predicted recall map is then increased linearly and capped at 0.35 by 70% of Stage-3 training. We keep the cap below one because the pseudo labels remain cleaner than early predictions, and completely replacing them can destabilize the action loss. At inference time, STAM always uses the predicted recall map.

For the Stage-3 objective in Eq. 5 of the main paper, we use $\lambda_\delta = 0.1$, Huber threshold $\beta_\delta = 0.1$, and

$$\phi(\delta) = \log(1 + \delta/\tau_{\text{unit}}), \qquad \tau_{\text{unit}} = 1\,\text{s}.$$

The transform keeps the zero target stable while preventing long recall times from dominating the regression loss. The effective recall map also uses the inference dead zone from the main method: predictions below $\epsilon_\delta = 0.05\,\text{s}$ are treated as no-memory patches.

We do not rely on action-loss gradients through discrete buffer lookup to train the recall map head. The primary recall map signal is the supervised Huber term against $\delta_t^\star$. This choice keeps the training target well defined even when retrieval uses nearest-older frame lookup, and avoids a common instability where the recall predictor collapses to a small set of easy but uninformative timestamps. The action loss still trains the memory-conditioned policy and the fusion pathway, while the recall map loss trains the policy to choose the memory used at inference.

C Evaluation Protocols

This appendix provides task definitions, success criteria, and aggregation rules for the experiments in Sec. 4 of the main paper.

C.1 Simulation Benchmarks

LIBERO. We use LIBERO as the current-observation control benchmark (Liu et al., 2023). It contains four standard suites: Spatial, Object, Goal, and Long. Spatial, Object, and Goal mainly test language-conditioned manipulation under changing object locations, object identities, and goal specifications. Long increases temporal length, but the benchmark is still used here primarily to check that a memory module does not damage the strong generalist skills already present in the VLA backbone. The metric is episode success rate. The LIBERO average in Table 1 of the main paper is the arithmetic mean over the four LIBERO suites, so each suite contributes equally to the reported average.

LIBERO-MEM. LIBERO-MEM is the object-centric memory benchmark introduced with Embodied-SlotSSM (Chung et al., 2025). It contains ten tasks built around four memory dimensions: Object Motion, Object Sequence, Object Relations, and Object Occlusion. Object Motion requires the policy to remember where an object has moved or how it has been manipulated. Object Sequence requires the policy to know which subgoal or repeated interaction has already occurred. Object Relations requires the policy to preserve relations among visually similar objects, such as which bowl was moved through which plate. Object Occlusion introduces cases where the relevant object or container state is no longer directly visible. These tasks are partially observable by construction: two frames can look similar while requiring different actions because the hidden object identity, relation, or subgoal progress differs. We report the LIBERO-MEM score as task progress or subgoal-completion percentage, matching the benchmark definition used by the source evaluation.

The single LIBERO-MEM value in Table 1 of the main paper is the benchmark-level average over the memory tasks.

MIKASA-Robo. MIKASA-Robo isolates atomic robotic memory skills in tabletop manipulation (Cherepanov et al., 2025). We evaluate the five VLA-facing tasks reported in Table 2 of the main paper. `ShellGameTouch` asks the robot to track which cup hides the target object after cup motion and then touch the correct cup; it mainly tests object-location memory. `InterceptMedium` asks the robot to infer the motion of a rolling ball from previous positions before aiming at the target; it tests spatial and velocity memory. `RememberColor3`, `RememberColor5`, and `RememberColor9` reveal a target cube color early in the episode and later require the robot to select the matching cube among three, five, or nine candidates; these tasks test object memory and memory capacity. Success is counted at the episode level according to the environment success condition. The average in Table 2 of the main paper is the arithmetic mean over the five reported MIKASA-Robo tasks. This benchmark is useful because failures are easy to interpret: a policy that can grasp and move objects but chooses the wrong cup or color has failed to preserve the relevant history, not merely failed at low-level control.

MemMimic. MemMimic evaluates memory policies under delayed, repeated, and trial-level dependencies (Gao et al., 2026). We report five simulation tasks in Table 3 of the main paper. `T1 Match Color` requires the robot to place an object according to an earlier color cue. `T1'` adds a random delay before the final color decision, increasing the gap between the cue and the action that depends on it. `T2 Place Back` requires the robot to return an object to the correct previous location. `T4 Iterative Pushing` asks the policy to adjust a push from previous trials, so the current action depends on remembered trial outcomes. `T5 Iterative Flinging` has the same trial-adaptation structure but with deformable-object dynamics. The metric is episode-level success rate. The reported average is computed over the available task columns in the table; missing entries are excluded rather than treated as failures. MemMimic therefore tests a different failure mode from MIKASA-Robo: the policy must retain and use history across a longer interaction process, not just remember one hidden attribute.

C.2 Real-World Robot Evaluation Protocol

The robot platforms and capability groups are introduced in Sec. 4.1 and Fig. 3(b) of the main paper. Below, we detail the individual tasks and scoring rules.

Figures S1 and S2 list the concrete Franka and DROID tasks used in the real-world evaluation. These task montages provide visual context for the capability grouping in this section: some tasks mainly test current-frame manipulation, while others require recalling target identity, occluded evidence, original state, count, or task progress.

General Manipulation. This group checks that adding memory does not break ordinary robot control. *Markov Manipulation* contains `Franka-Kitchen`, where the task is mainly decided by the current scene state. *Instruction Grounding* contains `Franka-Plush toy`, where the policy must follow a language instruction to select the intended target. *Actuation Precision* contains `Franka-Insert charger`, where success depends on precise execution. These tasks do not reward unnecessary history use; they are included because a memory module that improves non-Markovian tasks by sacrificing basic manipulation would not be a useful plug-in for generalist policies.

Memory Manipulation. This group contains tasks where the current frame can be ambiguous without history. *Target Recall* includes `ARX AC-1-Shell Game`, `Franka-Three cube`, and `DROID-Arrange cube`; the target identity or initial cue is visible earlier but must be recalled after movement, shuffling, or distraction. *Occlusion Recovery* uses `DROID-Put spoon`, where the policy must continue acting after the relevant local visual evidence has been occluded. *Origin Recall* uses `DROID-Cover pot`, where the robot must remember a previous object or container state after interaction. *Count Memory* uses `DROID-Add ingredients`, which requires remembering how many interactions have already been completed. *Progress Memory* includes `Franka-Close drawer`, `DROID-Put bowl + pick bread`, and `SO-101-Tic-Tac-Toe`; these tasks require the policy to preserve task progress across a longer sequence of actions. The memory group

is the primary real-world test of STAM's recall mechanism because the correct action depends on evidence that may no longer be visible.

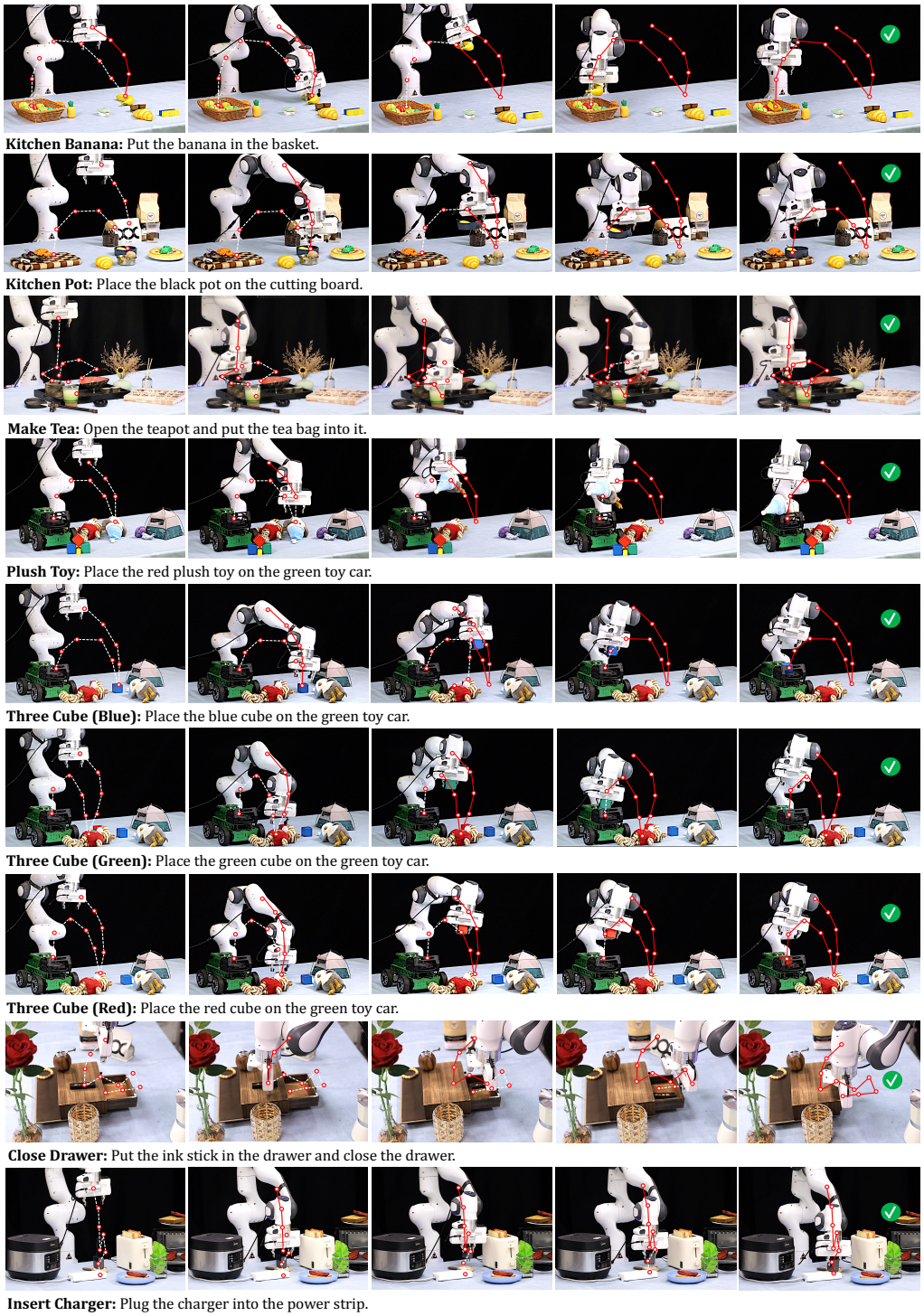


Figure S1: **Franka real-world tasks.** The montage shows all Franka tasks used in the real-world evaluation.

OOD Robustness. This group checks whether memory becomes harmful when history is unreliable or weakly related to the current decision. It contains `DROID-Flip bottle` and



Figure S2: **DROID real-world tasks.** The montage shows all DROID tasks used in the real-world evaluation.

`Franka-Make tea`. These tasks are not intended to show that every out-of-distribution perturbation is solved. Instead, they probe a specific risk of memory-augmented policies: if a model conditions on history unconditionally, shifted or irrelevant histories can distract it from the current observation. STAM is evaluated under the same task success criterion as the other real-world tasks, but the interpretation is different: stable performance here indicates that the learned recall map can keep memory closed or suppressed when historical evidence is not helpful.

For real-world aggregation, each concrete task is evaluated by final task success. Intermediate events such as grasping can be recorded for diagnosis, but group averages use final task success only. The Franka average excludes intermediate grasp columns and averages the final success columns. The DROID average is computed over the concrete DROID task columns; for the combined `Put bowl + pick bread` task, an episode is counted by the weaker of the two subtask outcomes so that both actions must succeed. The SO-101 and ARX AC-1 averages each contain one task in the current evaluation. The capability-group averages in Fig. 4 of the main paper are then computed from the task categories above, so each bar corresponds to a task-defined capability rather than a hardware-defined subset.

D Additional Results and Ablations

This appendix provides detailed simulation and real-world results underlying the main tables and figures. It also reports task-level ablations and additional evaluations of Stage-3 training and the SSMC architecture.

D.1 Full Simulation Results

Table S3 expands the LIBERO/LIBERO-MEM summary reported in Table 1 of the main paper. Benchmark definitions and scoring rules are provided in Appendix C.1.

Table S3: **LIBERO and LIBERO-Mem simulation results.** LIBERO entries are success rates. The LIBERO-Mem column reports task progress / subgoal completion percentage.

Model	LIBERO Spatial	LIBERO Object	LIBERO Goal	LIBERO Long	LIBERO Avg.	LIBERO Mem
OpenVLA-OFT (Kim et al., 2025)	97.6 ± 0.9	98.4 ± 0.8	97.9 ± 1.0	94.5 ± 1.3	97.1	–
GR00T N1.5 (NVIDIA GEAR, 2025)	98.2	99.4	97.2	87.8	95.7	–
π_0-FAST (Pertsch et al., 2025)	96.4 ± 0.7	96.8 ± 0.7	88.6 ± 1.0	60.2 ± 1.4	85.5	–
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	98.8 ± 0.2	98.2 ± 0.3	98.0 ± 0.9	92.4 ± 0.6	96.9	–
CronusVLA (Li et al., 2025)	97.3 ± 0.2	99.6 ± 0.0	96.9 ± 0.3	94.0 ± 1.0	97.0	–
MemoryVLA (Shi et al., 2026)	98.4	98.4	96.4	93.4	96.7	–
HAMLET (Koo et al., 2026)	99.0	100.0	99.2	92.2	97.6	–
ContextVLA (Jang et al., 2025)	98.4	99.0	97.2	93.4	97.0	–
π_0 (Black et al., 2024)	96.8 ± 0.8	98.8 ± 0.9	95.8 ± 1.1	85.2 ± 1.2	94.2	5.0
SlotVLA (Chung et al., 2025)	–	–	–	–	–	5.0
Naive E-SlotSSM (Chung et al., 2025)	89.0	91.3	83.0	54.3	79.4	14.8
Pi-Mem($\pi_{0.5}$)* (Torne et al., 2026)	94.3 ± 0.8	96.3 ± 0.6	95.3 ± 0.7	88.6 ± 1.2	93.6	17.3
Ours($\pi_{0.5}$ (Physical Intelligence et al., 2025))	98.8 ± 0.3	99.2 ± 0.2	99.2 ± 0.3	**95.6** ± 0.8	**98.2**	**32.4**
Ours(GR00T N1.5 (NVIDIA GEAR, 2025))	98.6 ± 0.4	99.4 ± 0.2	97.9 ± 0.5	90.3 ± 1.0	96.6	28.7

D.2 Full Real-World Task Results

Tables S4 and S5 give the task-level scores underlying the capability groups in Fig. 4 of the main paper. Each concrete real-world task is evaluated by final task success over 11 trials; therefore each task-level percentage lies on the $k/11$ grid after rounding. Franka grasp success is recorded as a diagnostic but is not used in Fig. 4 of the main paper, because the figure is meant to measure final task completion rather than intermediate contact.

Table S4: **Full real-world task-level results on Franka and SO-101.** Success rates are reported in percentage. The SO-101 column is a separate single-task setting and is not included in the Franka average.

Model	Kitchen	Three Cube	Plush Toy	Close Drawer	Insert Charger	Make Tea	Avg.	SO-101 Tic-Tac-Toe
π_0 (Black et al., 2024)	90.9	63.6	72.7	63.6	36.4	18.2	57.6	–
π_0-FAST (Pertsch et al., 2025)	27.3	27.3	45.5	36.4	0.0	0.0	22.8	–
GR00T N1.5 (NVIDIA GEAR, 2025)	100.0	63.6	81.8	72.7	36.4	27.3	63.6	36.4
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	100.0	63.6	81.8	81.8	36.4	36.4	66.7	45.5
Pi-Mem($\pi_{0.5}$)* (Torne et al., 2026)	90.9	72.7	90.9	90.9	36.4	27.3	68.2	54.5
Ours($\pi_{0.5}$ (Physical Intelligence et al., 2025))	**100.0**	**100.0**	**100.0**	**100.0**	**45.5**	**45.5**	**81.8**	**90.9**
Ours(GR00T N1.5 (NVIDIA GEAR, 2025))	**100.0**	90.9	90.9	**100.0**	**45.5**	36.4	77.3	72.7

Table S5: **Full real-world task-level results on DROID and ARX AC-1.** Success rates are reported in percentage. The ARX AC-1 Shell Game column is a separate single-task setting and is not included in the DROID average. Missing entries denote unevaluated robot-model combinations.

Model	Flip Bottle	Add Ingredients	Cover Pot	Put Spoon	Put Bowl + Bread	Arrange Cube	Avg.	ARX AC-1 Shell Game
OpenVLA-OFT (Kim et al., 2025)	36.4	54.5	81.8	63.6	72.7	27.3	56.1	–
π_0 (Black et al., 2024)	54.5	54.5	81.8	81.8	81.8	54.5	68.2	–
π_0-FAST (Pertsch et al., 2025)	45.5	72.7	81.8	81.8	90.9	63.6	72.7	–
GR00T N1.5 (NVIDIA GEAR, 2025)	27.3	54.5	81.8	63.6	72.7	45.5	57.6	0.0
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	54.5	72.7	90.9	81.8	81.8	63.6	74.2	0.0
Pi-Mem($\pi_{0.5}$)* (Torne et al., 2026)	45.5	72.7	90.9	81.8	81.8	72.7	74.2	0.0
Ours($\pi_{0.5}$ (Physical Intelligence et al., 2025))	**54.5**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**92.4**	**27.3**
Ours(GR00T N1.5 (NVIDIA GEAR, 2025))	36.4	81.8	**100.0**	90.9	**100.0**	81.8	81.8	**27.3**

Table S6 maps the platform-level tasks into the bars shown in Fig. 4 of the main paper. Task definitions and group aggregation rules are detailed in Appendix C.2.

Table S6: **Capability-group values underlying Fig. 4 of the main paper.** General, OOD, and Memory columns correspond to panels (a)–(c). The five memory-type columns correspond to the breakdown in panel (d). Missing entries are marked with "–".

Model	General Avg.	OOD Avg.	Memory Avg.	Target Recall	Occl. Recovery	Origin Recall	Count Memory	Progress Memory
OpenVLA-OFT (Kim et al., 2025)	–	36.4	60.0	27.3	63.6	81.8	54.5	72.7
π_0 (Black et al., 2024)	66.7	36.4	70.0	59.1	81.8	81.8	54.5	72.7
π_0-FAST (Pertsch et al., 2025)	24.3	22.8	69.1	45.5	81.8	81.8	72.7	63.7
GR00T N1.5 (NVIDIA GEAR, 2025)	72.7	27.3	59.4	36.4	63.6	81.8	54.5	60.6
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	72.7	45.5	71.5	42.4	81.8	90.9	72.7	69.7
Pi-Mem($\pi_{0.5}$)* (Torne et al., 2026)	72.7	36.4	73.9	48.5	81.8	90.9	72.7	75.7
Ours($\pi_{0.5}$ (Physical Intelligence et al., 2025))	**81.8**	**50.0**	**94.6**	**75.8**	**100.0**	**100.0**	**100.0**	**97.0**
Ours(GR00T N1.5 (NVIDIA GEAR, 2025))	78.8	36.4	86.1	66.7	90.9	**100.0**	81.8	90.9

D.3 Full Ablation Task Results

Table S7 gives the per-task results underlying Fig. 6(a) of the main paper. Removing fine memory produces the largest drop on each of the five MIKASA-Robo tasks, showing that the average decrease is not driven by a single task.

The left half of Table S8 reports the Flip Bottle and Add Ingredients results in Fig. 6(b)–(c) of the main paper. The right half gives the Stage-2 search-budget results on MemMimic T5 Iterative Flinging in panel (d). The main comparisons and budget choice are discussed in Sec. 4.4 of the main paper.

Stage-3 training and SSMC architecture. Table S9 evaluates the recall loss, confidence weighting, teacher forcing, and memory-core architecture. We replace the Huber loss for recall-map re-

Table S7: **MIKASA-Robo task-level ablations underlying Fig. 6(a) of the main paper.** All rows use STAM($\pi_{0.5}$) unless a memory component or recall rule is ablated. Average is the arithmetic mean over the five task columns.

Setting	ShellGame Touch	Intercept Medium	Remember Color3	Remember Color5	Remember Color9	Average
Ours($\pi_{0.5}$)	**96.0**	**88.0**	**87.0**	**73.0**	**20.0**	**72.8**
w/o Coarse Memory	91.0	84.0	82.0	67.0	18.0	68.4
w/o Fine Memory	24.0	31.0	27.0	20.0	6.0	21.6
Fixed-Window Recall	49.0	56.0	56.0	45.0	9.0	43.0
Discrete Recall	77.0	73.0	72.0	59.0	15.0	59.2
Global Recall Map	76.0	71.0	70.0	57.0	15.0	57.8

Table S8: **Additional ablation values underlying Fig. 6(b)–(d) of the main paper.** Left: real-world single-task ablations used for the OOD and Memory Manipulation bar panels. Right: Stage-2 candidate recall map budget on MemMimic `T5 Iterative Flinging`.

Setting	DROID Flip Bottle	DROID Add Ingredients
Ours($\pi_{0.5}$)	**54.5**	**100.0**
w/o Coarse Memory	45.5	90.9
w/o Fine Memory	45.5	27.3
Fixed-Window Recall	9.1	63.6
Discrete Recall	45.5	72.7
Global Recall Map	36.4	72.7

Budget N	Ours ($\pi_{0.5}$)	Ours (GR00T)
4	70.0	63.0
8	79.0	72.0
16	82.4	76.2
32	**83.5**	**77.3**
64	83.8	77.6
128	84.0	77.7

gression with L1 or MSE loss, or assign equal weight to all pseudo labels instead of using confidence weighting. The variant without teacher forcing uses predicted recall maps throughout Stage 3. For the memory-core ablations, we replace SSMC with an LSTM or Temporal Transformer. Every variant performs worse than complete STAM on all three evaluations, with the largest drop when confidence weighting is removed. These results support the training choices and memory-core architecture used in STAM.

Table S9: **Ablations of Stage-3 training and SSMC architecture.** LIBERO-MEM reports average subgoal completion; ShellGameTouch and InterceptMedium report task success rates. All values are percentages.

Setting	LIBERO-MEM Avg.	ShellGame Touch	Intercept Medium
STAM	**32.4**	**96.0**	**88.0**
L1 loss	29.3	89.0	82.0
MSE loss	27.1	86.0	79.0
w/o confidence weighting	9.4	33.0	29.0
w/o teacher forcing	21.3	83.0	72.0
LSTM	26.4	87.0	78.0
Temporal Transformer	25.6	87.0	74.0