REMEMBER LESS, ADAPT LONGER: BOUNDED-STATE IN-CONTEXT REINFORCEMENT LEARNING WITH RECURRENT SUMMARY MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

In-context reinforcement learning (ICRL) enables agents to adapt to new tasks without parameter updates, relying on memory to retain what they learn from experience. For agents operating continually, existing approaches using memory trade off bounded cost against access to past experience. Approaches using growing memories, such as full-history attention and accumulating summaries, incur storage and per-decision costs that increase with task length. Fixed-size memories bound storage but restrict access: sliding windows discard older steps, while recurrent networks compress even recent interactions into a hidden state. To avoid this trade-off, we introduce Recurrent Summary Memory (RSM), a Transformer policy combining exact attention over a short-term working buffer with a fixed-capacity long-term summary of earlier experience. When the buffer fills, RSM rewrites the summary from its previous contents and buffered interactions, then clears the buffer, keeping deployment memory and per-decision cost $O(1)$ in task length. On long-running partially observable benchmarks, frozen RSM policies sustain adaptation performance for eight times the training horizon using about 1% of full-history memory, and outperform all evaluated baselines at the final horizon, with a widening lead over growing memories. When the underlying task configuration changes mid-run, RSM adapts again and outperforms growing-memory and recurrent baselines. The policy relies on the summary: erasing it or blocking policy access removes its benefit, while swapping in another task's summary steers behaviour toward that task. Together, these results show that our fixed-capacity summary can retain task knowledge far beyond the training horizon.

1 INTRODUCTION

An agent placed in an unfamiliar environment often cannot directly observe everything it needs to act, so it must use past experience to inform its decisions and adapt to the environment (Kaelbling et al., 1998). In Dark Key-to-Door (Laskin et al., 2023), for example, an agent that searches a dark room for a hidden key and then a door should reuse the key's location on every later attempt; in a maze with unfamiliar controls, an inferred mapping from actions to movements should support navigation throughout the task. In-context reinforcement learning (ICRL) trains policies to adapt in this way: a single policy is trained across tasks to condition its decisions on its history of observations, actions, and rewards, enabling adaptation without test-time parameter updates (Laskin et al., 2023; Lee et al., 2023; Grigsby et al., 2024). Because its parameters remain fixed, memory must carry the task knowledge acquired during deployment (Duan et al., 2016; Wang et al., 2016). For an agent that keeps operating, continued adaptation therefore depends on preserving useful discoveries.

Existing memory architectures trade off bounded cost against access to experience. Full-history Transformers (Parisotto et al., 2020; Grigsby et al., 2024) let a policy revisit individual records, but their key/value caches and per-decision attention costs grow with task duration. Sliding windows bound these costs by discarding older records, even when their information remains useful (Xiao et al., 2024). Recurrent networks such as GRUs (Cho et al., 2014) maintain a fixed-size hidden state and provide strong baselines under partial observability, but compress even recent interactions (Heess et al., 2015; Ni et al., 2022; Morad et al., 2023). Summary memory offers a way to separate immediate access from longer-term retention: recent records provide direct context, while a

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.

compact representation carries earlier experience (Rae et al., 2020; Ge et al., 2024). Segment-level architectures implement this separation through local attention and representations carried between segments (Dai et al., 2019; Hutchins et al., 2022), but differ in what they retain. Recurrent Memory Transformers (RMTs) pass a fixed bank of memory tokens (Bulatov et al., 2022), whereas Memo appends one summary block per segment, so its state still grows (Gupta et al., 2025).

A fixed-capacity summary eliminates storage growth, but bounded storage alone does not guarantee sustained adaptation. Once a segment is discarded, its useful evidence must survive not only the next summary, but every subsequent rewrite until it is needed. Re-exploring the room may recover a forgotten key location, but not a cue presented only once (Ni et al., 2023). Conversely, when the hidden task changes, retaining obsolete evidence can obstruct adaptation. The summary must therefore learn both to preserve relevant information and to revise it. These demands extend beyond the training horizon, where longer deployments subject memory to more rewrites than it encountered during training. Success on finite training tasks need not imply reliable memory during continued operation; indeed, finite and continuing memory tasks can reverse the relative performance of recurrent and attention-based agents (Pleines et al., 2025). We therefore ask: *Can a Transformer policy sustain useful in-context adaptation beyond its local context and training horizon under a fixed deployment memory budget?*

We address this question with **Recurrent Summary Memory (RSM)**, building on segment-level recurrent memory (Bulatov et al., 2022). RSM combines exact attention over a short-term working buffer of the current segment's interactions with a fixed-capacity long-term summary. At each segment boundary, it writes a replacement summary from the previous summary and buffered steps, then clears the buffer. Only the new summary carries earlier experience into the next segment. In Key-to-Door, for example, recent movements remain directly accessible in the buffer, while discovered key and door locations must pass through the summary to guide later attempts. The summary persists across attempts within a task and resets between tasks. With fixed buffer and summary capacities, RSM requires $O(1)$ history-dependent deployment memory and per-decision computation in task length T, unlike full-history attention and accumulating summaries, whose state grows with T. Such bounds matter for robots operating on fixed onboard memory, game or simulation agents acting over thousands of steps, and services running many agents in parallel.

In our experiments, RSM uses 249.9 KB per agent and a 40-slot attention cache, compared with a 24.6 MB full-history cache after 4,000 steps, which grows by 6 KB per step to about 5 GB after a day of 10 Hz control, even for our small backbone. What the summary retains is learned from task reward, rather than determined by prescribed token-retention rules, as in StreamingLLM (Xiao et al., 2024), or auxiliary reconstruction losses, as in Compressive Transformers (Rae et al., 2020).

To test whether this bounded memory sustains adaptation, we evaluate RSM on four partially observable benchmarks: Dark Key-to-Door (location reuse), CountRecall (evidence accumulation; Morad et al., 2023), MazeRunner (navigation with unknown action mappings; Grigsby et al., 2024), and the passive T-Maze (one-time-cue recall; Ni et al., 2023). We compare against a full-history Transformer, a GRU, accumulating-summary Memo, and a sliding window of the same state, trained with the same learner, inputs, losses, and interaction budget. At the training horizon, no baseline leads RSM by more than a practical margin. As deployments need not end where training tasks do, frozen policies continue acting for eight times their training horizon: RSM sustains adapted performance while growing memories deteriorate. It also re-adapts when the hidden layout or maze changes mid-run, as the environment of a long-running agent eventually will. Our main contributions are:

- **Bounded summary memory for online in-context RL.** RSM brings the fixed memory bank of Recurrent Memory Transformers from language modelling to online in-context RL: a single summary, overwritten at each segment boundary beside an exact working buffer and trained with off-policy RL losses alone through every carried write. Unlike Memo, its deployment state and per-decision cost remain $O(1)$ in task length (Section 3).
- **Sustained adaptation beyond the training horizon.** Frozen RSM policies maintain adapted performance over eight times the training horizon and re-adapt when the hidden task changes, whereas growing memories deteriorate (Section 4.2).
- **Evidence that decisions depend on summary content.** Clearing the summary, transplanting another task's summary into it, and blocking the policy's access to it show that the policy uses its content; linear probes recover hidden task variables; and write-rule ablations show greater stability with overwriting than with additive updates (Sections 4.3–4.4).

2 Preliminaries and Problem Formulation

Task setting and in-context adaptation. A task ξ, such as a room layout or an action mapping, defines a finite-horizon POMDP $\mathcal{M}_\xi = (\mathcal{X}, \mathcal{O}, \mathcal{A}, P_\xi, \Omega_\xi, R_\xi, \rho_\xi, T_\xi)$ (Kaelbling et al., 1998), with hidden states $x_t \in \mathcal{X}$, observations $o_t \in \mathcal{O}$ drawn from the observation function $\Omega_\xi(\cdot \mid x_t)$, actions $a_t \in \mathcal{A}$, dynamics P_ξ, reward function R_ξ, initial-state distribution ρ_ξ and horizon T_ξ. The agent plays each task as one *outer task*, an episode of $\mathcal{M}_\xi$ that may contain several physical attempts; an attempt reset is a transition of P_ξ back to a start state under the same ξ. At decision t the policy conditions on the history $\mathcal{H}_t = (o_0, a_0, r_1, o_1, \ldots, a_{t-1}, r_t, o_t)$, where $r_{t+1} = R_\xi(x_t, a_t)$ is the environment's native (unscaled) reward. Every interface call is a step, including the reset-only call that starts each new attempt: it takes no physical action, so the environment ignores a_t, returns the new attempt's first observation and gives no reward ($r_{t+1} = 0$), yet it counts toward the task length and the training budget. ICRL learns one set of parameters θ for the policy and its encoders, shared across all training tasks, so that the policy adapts to new tasks through interaction (Duan et al., 2016; Wang et al., 2016; Grigsby et al., 2024). It maximizes the expected discounted return $J_\gamma(\theta)$ over training tasks $\xi \sim p_{\text{train}}$ and the outer-task trajectories τ that π_θ generates in $\mathcal{M}_\xi$, with discount $\gamma \in (0, 1]$:

$$J_\gamma(\theta) = \mathbb{E}_{\xi \sim p_{\text{train}},\, \tau \sim \pi_\theta, \mathcal{M}_\xi} \left[\sum_{t=0}^{T_\xi - 1} \gamma^t r_{t+1} \right]. \tag{1}$$

Because the return sums the rewards of every attempt in an outer task, information gathered in early attempts pays off in later ones. At evaluation, θ is frozen and the policy runs on held-out tasks disjoint from training; we report native, undiscounted metrics. More details in Appendix C.

History-dependent policies. All compared policies receive the same public input, the current observation with the native RL^2 action–reward feedback (Duan et al., 2016); task identifiers, hidden layouts and evaluator labels never enter it. A stateless timestep encoder f_ϕ embeds each step's record u_t (with a_{-1} and r_0 set to zero), and a causal history encoder F_θ updates the policy state:

$$u_t = (o_t, a_{t-1}, r_t), \qquad e_t = f_\phi(u_t), \qquad (m_t, s_t) = F_\theta(s_{t-1}, e_t), \qquad a_t \sim \pi_\theta(\cdot \mid m_t). \tag{2}$$

Here, s_t denotes all history-dependent state that the policy retains for future decisions, m_t the actor–critic representation and s_{-1} a task-independent initial state; θ includes ϕ. The state s_t is updated at every step and kept across attempt resets, and returns to s_{-1} only when a new outer task begins.

Bounded-state adaptation objective. We seek a policy that adapts through interaction while its history-dependent deployment state respects a fixed budget B:

$$\max_\theta J_\gamma(\theta) \quad \text{subject to} \quad \text{bytes}(s_t) \le B \quad \text{for all } \xi,\ \mathcal{H}_t,\ 0 \le t < T_\xi, \tag{3}$$

where $\text{bytes}(s_t)$ is the memory needed to store s_t at implementation precision (its summary or hidden-state tensors, key/value caches and counters, but not the parameters, workspace or training resources) and B does not depend on T_ξ: the deployment state must be $O(1)$ in the task length, whereas a full-history Transformer, whose cache holds every record, needs $O(T_\xi)$. More details in Appendices D.1 and D.2.

3 Recurrent Summary Memory

RSM (Figure 1) builds on segment-level recurrent memory (Bulatov et al., 2022) and splits a task into segments of $C = 32$ steps. Within a segment, each decision attends exactly to the segment's steps so far, the *working buffer*, and to a *summary* of $M = 4$ learned vectors (tokens) carried over from earlier segments. When the segment ends, M write queries read the old summary and the whole buffer, and their outputs replace the summary; the buffer is then emptied. The agent therefore never holds more than 32 recent steps and 4 summary tokens, however long the task runs. Section 3.1 defines this layout, Section 3.2 the overwrite and its training, and Section 3.3 the bounds.

3.1 Working buffer and recurrent summary

We partition the per-step embeddings $e_t = f_\phi(u_t)$ of Equation 2 into consecutive segments of $C = 32$ records: record t belongs to segment $b = \lfloor t/C \rfloor$ at position $i = t - bC + 1$, so that $e_{b,i} = e_t$

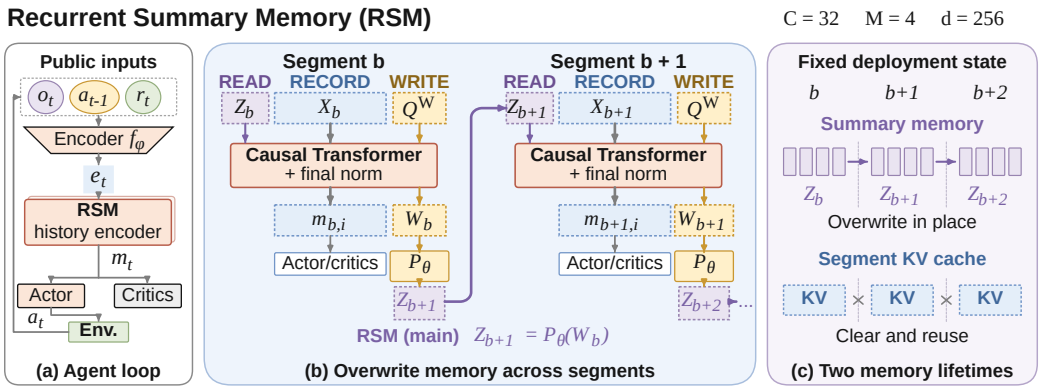


Figure 1: **Recurrent Summary Memory**. (a) The stateless encoder f_ϕ maps each public record to e_t; RSM supplies m_t to the actor and critics. (b) $X_b = (e_{b,1}, \ldots, e_{b,n_b})$ contains the segment's records. Each RECORD output $m_{b,i}$ uses Z_b and the causal prefix $X_{b,1:i}$; normalized WRITE outputs W_b yield $Z_{b+1} = P_\theta(W_b)$ for the next segment. Parameters are shared across segments; token embeddings are omitted. (c) Successive snapshots of the same bounded deployment state: the summary is replaced and the segment KV cache cleared (crosses). Counters are also retained; full-task training memory grows with task length.

and $m_{b,i} = m_t$. Segment b receives a summary $Z_b \in \mathbb{R}^{M\times d}$ of $M = 4$ tokens (its rows) and the records $X_b = (e_{b,1}, \ldots, e_{b,n_b})$, where $1 \le n_b \le C$ and only the last segment of a task can have $n_b < C$. One causal Transformer (Vaswani et al., 2017) of width $d = 256$, with $L = 3$ layers and eight heads, supplies both the decision representation and the next summary:

$$m_{b,i} = \mathcal{R}_\theta(Z_b, X_{b,1:i}), \qquad Z_{b+1} = \mathcal{U}_\theta(Z_b, X_b), \qquad Z_0 = Z_{\text{init}}, \tag{4}$$

where Z_{init} is a learned, task-independent initial summary. Both maps share the slot layout

$$[\ \textsc{Read}(Z_b) \mid \textsc{Record}(X_b) \mid \textsc{Write}(Q^{\text{W}})\ ], \tag{5}$$

where $Q^{\text{W}} \in \mathbb{R}^{M\times d}$ is a learned bank of write queries, and READ, RECORD and WRITE embed their arguments as rows of that role, with role, memory-slot and segment-local position embeddings and, for records, an input projection. The working buffer is the per-layer key/value cache of these $M+C+M$ slots. All layers apply ordinary causal attention over this slot order. The final-normalized RECORD and WRITE outputs are the decision representations $m_{b,i}$ and the rows of $W_b \in \mathbb{R}^{M\times d}$. Because the WRITE slots follow every RECORD slot, W_b cannot affect the current segment's decisions and reaches the policy only through Z_{b+1}; READ and WRITE slots are not environment steps and receive no direct RL loss. More details in Appendices A.3 and B.

3.2 Overwrite Update and Training

At each boundary RSM replaces the carried summary with the projected WRITE outputs; the residual-write ablation RSM-R adds them instead:

$$Z_{b+1} = P_\theta(W_b) \quad \text{(RSM)}, \qquad Z_{b+1} = Z_b + P_\theta(W_b) \quad \text{(RSM-R)}, \tag{6}$$

where P_θ applies one affine map $\mathbb{R}^d \to \mathbb{R}^d$ to each of the M rows, so that $\mathcal{U}_\theta(Z_b, X_b) = P_\theta(W_b)$ for RSM. The WRITE outputs depend on Z_b through the READ rows, which makes the overwrite recurrent without an additive identity path between summaries: the writer must express both the retained information and the new evidence in M output tokens. Only Z_{b+1} crosses a boundary: the working buffer is cleared and the slot counter restarts. A write occurs only when segment b is full and a further record arrives, so the last segment of a task is never written.

RSM is trained with AMAGO's off-policy actor–critic learner (Grigsby et al., 2024), shared by all compared policies: each update replays complete outer tasks, recomputes their summaries from Z_{init} under the current weights and backpropagates through every carried write (Werbos, 1990), with no summary target or auxiliary memory loss. Every later loss therefore reaches every earlier write through $Z_{b+1} = P_\theta(W_b)$; under the overwrite, the Jacobian $\partial Z_{b+2}/\partial Z_{b+1}$ passes only through attention to the READ rows, whereas RSM-R adds an identity term. The states of the parallel agents are rebuilt from their public prefixes after each update. More details in Appendices B and C.

3.3 Deployment-state and summary bounds

Proposition 1 (Bounded deployment state)**.** *For fixed C, M, d and L, each agent of a frozen RSM policy retains its summary (Md values), the working buffer's per-layer key/value caches ($2L(C + 2M)d$ values), a 32-bit occupied-slot counter and a 64-bit segment counter. With its tensors in 32-bit floating point (FP32), this state occupies*

$$B_{\text{deploy}} = 4\big[Md + 2L(C + 2M)d\big] + 12 \quad \textit{bytes}, \tag{7}$$

independent of task duration, for RSM-R and RSM w/o memory alike.

Space complexity. With the values of Section 3.1, $B_{\text{deploy}} = 249{,}868$ bytes per agent, so RSM's deployment state is $O(1)$ in the task length T_ξ, as for a GRU (Cho et al., 2014), whereas a full-history Transformer of the same L and d caches keys and values for every record ($2LT_\xi d$ values) and Memo (Gupta et al., 2025) for M summary tokens per segment (about $2LMT_\xi d/C$ values), both $O(T_\xi)$. Every RSM query also attends to at most $C + 2M = 40$ cached slots, so attention per decision is $O(1)$ in T_ξ as well. The overwrite also keeps every summary row within a fixed bound at any horizon, whereas the residual write admits only a bound that grows with every write (Proposition 2). More details in Appendix D.

4 Experiments

We ask (1) whether RSM adapts as well as memories that grow with the task, at the training horizon and far beyond it (Section 4.2); (2) whether its decisions use the summary (Section 4.3); and (3) which design choices matter (Section 4.4).

4.1 Experimental setup

Environments. We use four partially observable benchmarks. In **Dark Key-to-Door** (Key-to-Door below; Laskin et al., 2023; Grigsby et al., 2024), the agent searches an 8×8 room for a hidden key and door over repeated attempts in one 500-step task; **CountRecall** (POPGym; Morad et al., 2023) deals 104 cards from four categories and asks at every step how many of a queried category have been dealt; in **MazeRunner** (Grigsby et al., 2024), the agent reaches three ordered goals in a hidden 15×15 maze with permuted actions in one 500-step attempt; and the **passive T-Maze** (Ni et al., 2023) requires recalling a cue shown only at the first step of a 128-cell corridor.

Baselines and Ablations. All policies share the stateless timestep encoder, the public inputs and the AMAGO off-policy learner; the Transformers share width 256, three layers and eight heads. RSM uses a working buffer of 32 records and $M = 4$ summary tokens (Section 3). We compare it with the **full-history Transformer**, whose key–value cache grows with every record; AMAGO's recurrent **GRU**, whose state is fixed; and **Memo** (Gupta et al., 2025), which appends four summary tokens per 32-record segment, both as published (jittered segment lengths) and with **fixed segments**. We call the full-history Transformer and Memo the *growing memories*. A further baseline, the **sliding window**, is a Transformer without a summary that attends only to the last $W = 40$ records at every layer, the largest window within RSM's per-agent state. The ablations train **RSM w/o memory**, which resets the summary at every segment boundary; **RSM-R**, which adds the write to the carried summary instead of overwriting it (Equation 6); RSM with $M \in \{1, 4, 8, 16\}$ summary tokens on Key-to-Door; and, on CountRecall, **RSM detach**, whose summary is detached at every boundary so that no later loss reaches the write.

Evaluation Protocol. Every policy is trained three times, with seeds 42, 100 and 2026, for 8M steps each, and evaluated at its final checkpoint by greedy rollouts on 256 held-out *confirmation* tasks per benchmark, whose layouts, card streams and mazes never occur in training. The metrics are doors per 500 steps on Key-to-Door, exact accuracy on CountRecall (*CR evicted*: queries whose evidence has left the buffer), the fraction of the three goals reached on MazeRunner, and success on the T-Maze, each first at the *training horizon*, the length of a training task. CountRecall and the T-Maze have fixed-length tasks; Key-to-Door and MazeRunner also run for 4,000 steps, eight times that horizon, counting successes per 500 steps: in the *same task* the layout or map stays fixed while attempts continue, and in a *new task* a new layout arrives every 500 steps, or a new maze with the same action mapping every 1,000, with memory carried. We report means over tasks and seeds and

Table 1: **Performance at the training horizon and over long tasks** (settings in Section 4.1). Training horizon: Key-to-Door (KD) doors per 500 steps, CountRecall (CR) exact accuracy, MazeRunner (MR) fraction of goals reached, T-Maze (TM) success. Long-running: doors or goals per 500 steps, in the last 500 of 4,000 steps (same task), per later layout or on the fourth map (new task). CR evicted: accuracy on evidence that left the buffer. State: peak per agent. Bold: best compared memory.

	Training horizon				Long-running, 4,000 steps					
					same task		new task			
	KD ↑	CR ↑	MR ↑	TM ↑	KD ↑	MR ↑	KD ↑	MR ↑	CR evicted ↑	State ↓
RSM (ours)	29.1	0.989	0.804	**1.000**	**46.0**	**26.6**	**9.4**	**23.0**	0.986	249.9 KB
Full-history Transformer	27.4	**0.998**	0.737	**1.000**	28.2	0.8	0.8	0.4	**0.998**	24.58 MB
GRU	**29.6**	0.930	0.643	**1.000**	40.6	19.1	1.6	16.1	0.937	**3.1 KB**
Memo	**29.6**	0.956	**0.863**	**1.000**	10.6	1.8	0.5	2.0	0.939	3.24 MB
Memo, fixed segments	26.8	0.944	0.836	**1.000**	31.4	2.8	0.6	2.9	0.933	3.24 MB
RSM w/o memory	2.0	0.680	0.738	0.514	2.9	19.2	1.9	17.9	0.538	249.9 KB
RSM-R (residual)	29.8	0.990	0.756	1.000	45.2	5.0	1.8	0.4	0.988	249.9 KB

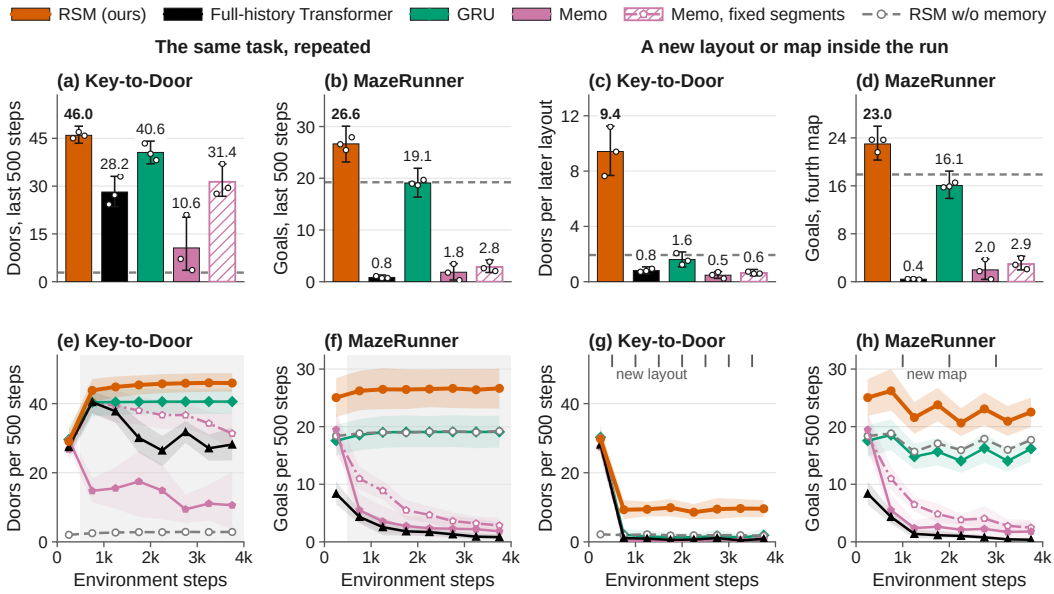


Figure 2: **Long-running tasks** (4,000 steps). Columns: the same layout or map throughout, or a new one at intervals. (a–d) Final-window scores, doors per later layout, or goals on the fourth map; dots: seeds; dashed: RSM w/o memory. (e–h) Scores per 500-step window; shading marks steps beyond training, ticks mark task changes.

compare policies by paired differences on the same tasks and seeds: a difference is *consistent* when it exceeds a practical margin δ (1 door, 0.05 accuracy, 0.1 goal fraction or one goal per 500 steps, and 0.25 success), its 95% joint seed/task bootstrap interval (Agarwal et al., 2021) excludes zero and all three seeds agree in sign, and *equivalent* when the whole interval lies within $\pm\delta$; any other difference is *no consistent difference*. Bands and whiskers are 95% intervals; dots mark seeds. More details in Appendices A, E and F.7.

4.2 Adaptation at and beyond the training horizon

At the training horizon. No compared memory is consistently better than RSM at the training horizon, and RSM pulls ahead of them over long tasks (below). It is equivalent to the full-history Transformer on CountRecall, MazeRunner and the T-Maze, consistently better than the GRU on CountRecall and MazeRunner, and behind Memo on MazeRunner on every seed, by a mean gap below δ; on Key-to-Door the intervals are too wide to separate it from the growing memories and the GRU (Tables 1 and 7). These scores reflect adaptation within a task: on Key-to-Door, door success rises from about 0.15 on the first attempt to above 0.8 on the ninth for every memory except

RSM w/o memory (Appendix F.1). Its state stays fixed while the full-history cache and Memo's store grow with every step; after 4,000 steps it is about a hundredth of the full-history cache (Figure 3).

Over a long task. In the same task (Table 1), RSM and the GRU sustain their adapted performance over the whole run, RSM at a higher level and, in the last window, ahead of every compared memory on every seed (Figure 2a,b,e,f; Table 1), so its advantage widens as the task runs on. The growing memories fall from the rate they reach early in the run, Memo most steeply, and on the repeated MazeRunner map they collapse towards zero goals; they also meet longer contexts than in training, while RSM's context stays at 40 slots (Section 3.3).

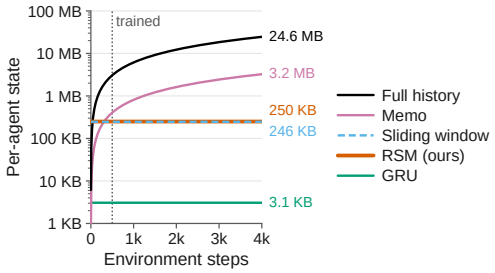


Figure 3: **Per-agent state** against task length.

A window of the same state. The sliding window, with about the same state as RSM, trails it consistently on Key-to-Door at the training horizon, where every position was seen in training, so evidence older than 40 records matters; on MazeRunner the two show no consistent difference. Over 4,000 steps the window falls to a few doors or goals per window (Table 2), but it then also meets untrained positions, so this comparison is operational rather than a clean test of summarisation.

Table 2: **A window of the same state** (the last 40 records per layer, 246.1 KB per agent).

	Training horizon		Last 500 of 4,000	
	KD	MR	KD	MR
RSM	29.1	0.804	46.0	26.6
window	24.1	0.750	4.9	2.6

After the task changes. In the new task (Table 1; Figure 2c,d,g,h), RSM drops after each change and recovers once it finds the new layout or map, largely on MazeRunner and in about a third of Key-to-Door layouts, yet stays above every other memory on every later layout, map and seed. On Key-to-Door the others average under two doors per layout; on MazeRunner the GRU and RSM w/o memory stay lower and the growing memories collapse. More details in Appendices E.2 and F.2.

4.3 Do the decisions use the summary?

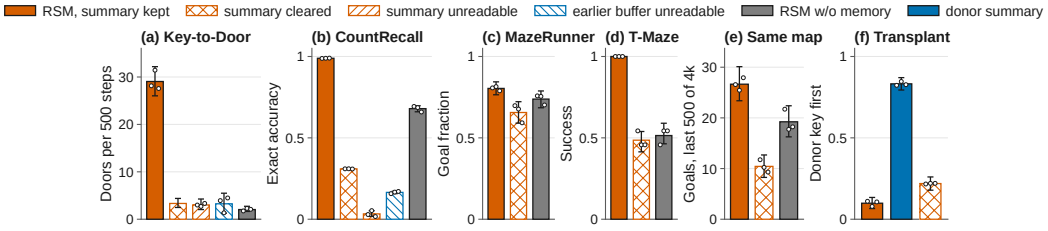


Figure 4: **Interventions on the frozen RSM policy.** (a–e) The summary kept or cleared at every boundary, beside RSM w/o memory, and in (a,b) the summary or the earlier buffer unreadable to the decisions ($\beta = \infty$); (e) MazeRunner goals in the last 500 of 4,000 steps on one map. (f) Key-to-Door tasks whose next attempt visits the donor's key first, after a transplant. Dots: seeds.

Clearing, transplanting and blocking. We test causally whether the decisions depend on the carried summary (Figure 4). Resetting RSM's summary to Z_{init} at every boundary, with the working buffer intact, removes the benefit of the summary on every benchmark and every seed, and leaves the policy well below RSM w/o memory on CountRecall and the long MazeRunner task and at its fixed-turn level on the T-Maze: the trained policy does not fall back on the buffer-only strategy that RSM w/o memory learns (Figure 4a–e). To transplant, we replace each task's summary at one boundary with the summary that a different task, the *donor*, carried there; behaviour then follows the donor task: on Key-to-Door the next attempt visits the donor's key cell first in most tasks (Figure 4f), and on CountRecall the policy answers with the donor's counts until the next rewrite. As attention weights alone do not explain a decision (Jain & Wallace, 2019), we also intervene on them: adding $-\beta$ to every decision's attention scores on the four summary keys leaves the summary carried but unreadable at $\beta = \infty$, which reduces RSM to the level of RSM w/o memory on Key-to-Door

and below it on CountRecall (Figure 4a,b), with a loss growing with β. Blocking the earlier buffer records costs about as much: the decisions depend on both memories.

Retention and Attention. On CountRecall, we group queries by how often the summary has been rewritten since the queried category was first dealt (Figure 5). Queries whose evidence has passed through up to three rewrites are answered about as accurately as by the full-history Transformer, which keeps every card, while the GRU and both Memo variants are less accurate in the last segment. A linear read-out (ridge regression) from the summary recovers every CountRecall count almost perfectly and the hidden key's position on Key-to-Door well, on every seed: four tokens hold the task's hidden variables in a readable form. The decisions also attend to the summary far more than the same slots in RSM w/o memory; on Key-to-Door this extra attention is flat across the room on the first attempt, while the agent searches, and on later attempts is higher in the room's interior than along its walls (Figure 6): once the agent has something to recall, how much it reads the summary varies slightly with where it is. More details in Appendices F.3–F.5.

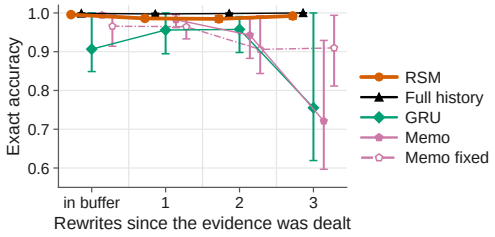


Figure 5: **CountRecall retention**: accuracy by the number of summary rewrites since the queried evidence was dealt.

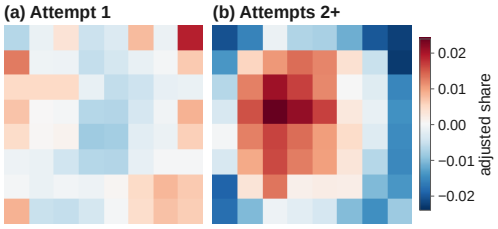


Figure 6: **Where the summary is read** in the Key-to-Door room: summary attention adjusted for segment position, by the agent's cell.

4.4 Ablations

Carrying and training the summary. Removing the carried summary and keeping only the 32-record buffer removes most of the adaptation (Table 1, RSM w/o memory), consistently on every seed on Key-to-Door, CountRecall and the T-Maze, and the loss widens over the 4,000-step Key-to-Door task. On MazeRunner a single 500-step attempt can largely be solved from the buffer, so the summary adds less than δ at the training horizon, but several goals per window over longer tasks. How the write is trained also matters: detaching the summary at every boundary, so that no later loss reaches the write, lowers RSM's CountRecall accuracy on every seed (0.914 against 0.989), yet keeps most of the gain over RSM w/o memory (0.680). Training each write through the decisions that later read it thus improves what the summary keeps.

Overwrite versus additive write. RSM-R, which adds the write instead of overwriting, stays within δ of RSM in mean at the training horizon and over the 4,000-step Key-to-Door task, but over many MazeRunner attempts on one map it declines until it falls below RSM w/o memory and on the map-change task it collapses by the fourth map, on every seed, while after a Key-to-Door layout change it adapts again no better than RSM w/o memory (Table 1). Wiping its memory at every attempt boundary preserves its rate, so the decline comes from what it carries. Its summaries drift in one direction until the first and the last are anti-correlated, at a norm tens of times larger than RSM's: a drift that Proposition 2 bounds for the overwrite but not for the additive write.

Summary length. Varying the number of summary tokens from one to sixteen on Key-to-Door, with buffer, backbone and budget fixed, changes adaptation by no consistent margin (Table 3): the paired differences from four tokens change sign across seeds at the training horizon, and in the last window every length completes 45 to 46 doors. Even a single token shows no consistent difference from four, and a larger summary, at a larger state, buys no further adaptation on this task. More details in Appendices F.4 and F.6.

Table 3: **Summary length on Key-to-Door** (4,000-step tasks; 95% intervals).

	$M = 1$	$M = 4$ (RSM)	$M = 8$	$M = 16$
State per agent	209.9 KB	249.9 KB	303.1 KB	409.6 KB
Doors, steps 1–500	29.6 [26.5, 33.0]	29.1 [25.9, 32.4]	30.8 [28.5, 33.1]	30.1 [27.8, 32.4]
Doors, steps 3,501–4,000	45.3 [42.2, 48.2]	46.0 [43.3, 48.6]	45.3 [42.5, 48.1]	45.9 [43.1, 48.7]

5 Related Work

Beyond the memory designs compared in Section 1, RSM relates to two lines of work: in-context reinforcement learning, which asks an agent to adapt from its own history, and memory for sequential control, which determines how much of that history the agent can keep.

In-context reinforcement learning. In-context RL trains one policy across many tasks so that it adapts to a new task from its own history, without updating its parameters. Recurrent meta-RL first achieved this through hidden-state updates (Duan et al., 2016; Wang et al., 2016), and PEARL and VariBAD infer a belief over the task from context (Rakelly et al., 2019; Zintgraf et al., 2020). Sequence models extend the idea to longer histories: Algorithm Distillation imitates the learning histories of RL algorithms (Laskin et al., 2023), Decision-Pretrained Transformers predict optimal actions from interaction data (Lee et al., 2023), and AMAGO and ReLIC learn in context through off-policy and on-policy updates over long sequences (Grigsby et al., 2024; Elawady et al., 2024). How well such agents remember is harder to measure: memory is easily confounded with exploration and credit assignment (Pasukonis et al., 2023; Ni et al., 2023; Cherepanov et al., 2026a), finite and endless versions of a task can reverse the ranking of recurrent and Transformer memories (Pleines et al., 2025), and retaining information differs from rewriting it when the task changes (Shchendrigin et al., 2026). Our evaluation follows these findings: we ran frozen policies far beyond the training horizon, changed the hidden task mid-run, and intervened on the summary directly.

Memory for sequence-based control. Like recurrent networks, state-space models and gated linear attention keep a fixed-size state (Lu et al., 2023; Pramanik et al., 2024), and Stable Hadamard Memory calibrates each update to erase what is no longer needed (Le et al., 2025). Learned summaries compress long contexts in language models (Chevalier et al., 2023; Ge et al., 2024), sometimes with predictive or reconstruction losses (Wayne et al., 2018; Rae et al., 2020). In RL, RATE and ELMUR carry segment memory offline, ELMUR detaching it between segments (Cherepanov et al., 2026c;b), and Memo, the closest online method, compresses each segment into summary tokens and backpropagates through them, but keeps every summary block, so its state grows with the task (Gupta et al., 2025). RSM builds on Recurrent Memory Transformers (Bulatov et al., 2022), which rewrite a fixed bank of memory tokens under a language-modelling loss, and on AMAGO's learner and benchmarks (Grigsby et al., 2024); it adds a bounded memory trained online from task reward alone and its evaluation beyond the training horizon. Like Memo, it trains its summary through later decisions, but keeps only a working buffer and a single summary, overwritten at every boundary.

6 Discussion and Conclusion

We introduced Recurrent Summary Memory, a Transformer policy that attends exactly to a short working buffer and carries a fixed-capacity summary, overwritten at every segment boundary and learned from task reward alone. With a deployment state fixed at 249.9 KB per agent, about a hundredth of a full-history cache after 4,000 steps, RSM is not consistently outperformed by memories that grow with the task at the training horizon. Over tasks eight times longer, it keeps the performance it reached by adapting and adapts again when the hidden task changes, ending ahead of every compared memory, whereas the growing memories deteriorate and a sliding window of the same state falls behind. Clearing, transplanting and blocking the summary show that the decisions rely on what it carries, and the overwrite keeps the summary bounded and stable where an additive write drifts. A memory of fixed size can thus keep an agent adapting far beyond its training horizon, provided it learns what to keep at every rewrite.

Three limits remain. Our training tasks were at most 500 steps long; only the deployment memory is bounded, since each training update replays whole tasks and backpropagates through every summary write; and on Key-to-Door, RSM learns more slowly than every other memory that learns the task. Future work can bound training memory as well, for example by backpropagating through only the most recent summaries: on CountRecall, cutting the gradient at every boundary still kept most of the summary's benefit (Section 4.4). Update rules could also learn when to keep and when to revise evidence over far longer horizons, and RSM could be carried to visual and embodied agents, such as robots that run continuously for hours on fixed onboard memory.

AI USE STATEMENT

In this work, we used generative AI tools to assist with developing and executing training and evaluation scripts. We did not use generative AI tools to generate research ideas; the research ideas were developed independently by the authors. Additionally, we used generative AI tools to polish the manuscript for clarity, concision, and readability. We have reviewed all AI-assisted work, including the manuscript revisions and the training and evaluation scripts. We take responsibility for the final content of this work, including all text, code, claims, and experimental results produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Section 3 defines the memory operator, and Appendices B–D provide pseudocode, the learning objective and the memory analysis. Appendix A reports the configurations, training budgets, task splits and evaluation procedures. An anonymized repository containing the implementation, pinned environment, experiment configurations, tests, summary evaluation records, and scripts and notebooks used to regenerate the reported figures and tables is available at https://anonymous.4open.science/r/RSM-ICLR-6AF7/README.md.

REFERENCES

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville, and Marc Bellemare. Deep Reinforcement Learning at the Edge of the Statistical Precipice, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/f514cec81cb148559cf475e7426eed5e-Abstract.html.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. Recurrent memory transformer. *Advances in neural information processing systems*, 35:11079–11091, 2022.

Egor Cherepanov, Nikita Kachaev, Artem Zholus, Alexey K. Kovalev, and Aleksandr I. Panov. Unraveling the Complexity of Memory in RL Agents: an Approach for Classification and Evaluation. *arXiv preprint arXiv:2412.06531*, 2026a. URL https://arxiv.org/abs/2412.06531.

Egor Cherepanov, Alexey K. Kovalev, and Aleksandr I. Panov. ELMUR: External Layer Memory with Update/Rewrite for Long-Horizon RL Problems. *arXiv preprint arXiv:2510.07151*, 2026b. URL https://arxiv.org/abs/2510.07151.

Egor Cherepanov, Aleksei Staroverov, Alexey Kovalev, and Aleksandr Panov. Recurrent action transformer with memory. In *International Conference on Learning Representations*, volume 2026, pp. 129379–129407, 2026c.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846, 2023.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734. Association for Computational Linguistics, 2014. doi: 10.3115/v1/D14-1179.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 2978–2988, 2019.

Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Ahmad Elawady, Gunjan Chhablani, Ram Ramrakhya, Karmesh Yadav, Dhruv Batra, Zsolt Kira, and Andrew Szot. ReLIC: A recipe for 64k steps of in-context reinforcement learning for embodied AI. *arXiv preprint arXiv:2410.02751*, 2024.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context Autoencoder for Context Compression in a Large Language Model. In *International Conference on Learning Representations*, volume 2024, pp. 2591–2607, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/0b276510ec2d3f6613a8b60c41ff0438-Paper-Conference.pdf.

Jake Grigsby, Linxi Fan, and Yuke Zhu. AMAGO: Scalable in-context reinforcement learning for adaptive agents. In *International Conference on Learning Representations*, volume 2024, pp. 26919–26952, 2024.

Gunshi Gupta, Karmesh Yadav, Zsolt Kira, Yarin Gal, and Rahaf Aljundi. Memo: Training memory-efficient embodied agents with reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 104392–104414, 2025. doi: 10.52202/085713-3486.

Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015. URL https://arxiv.org/abs/1512.04455.

DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. Block-Recurrent Transformers. In *Advances in Neural Information Processing Systems*, volume 35, pp. 33248–33261, 2022. doi: 10.52202/068431-2409. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/d6e0bbb9fc3f4c10950052ec2359355c-Paper-Conference.pdf.

Sarthak Jain and Byron C. Wallace. Attention is not explanation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 3543–3556. Association for Computational Linguistics, 2019. doi: 10.18653/v1/N19-1357.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998. doi: 10.1016/s0004-3702(98)00023-x.

Steven Kapturowski, Georg Ostrovski, John Quan, Rémi Munos, and Will Dabney. Recurrent Experience Replay in Distributed Reinforcement Learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=r1lyTjAqYX.

Michael Laskin, Luyu Wang, Junhyuk Oh, Emilio Parisotto, Stephen Spencer, Richie Steigerwald, DJ Strouse, Steven Stenberg Hansen, Angelos Filos, Ethan Brooks, Maxime Gazeau, Himanshu Sahni, Satinder Singh, and Volodymyr Mnih. In-context reinforcement learning with algorithm distillation. In *International Conference on Learning Representations*, 2023.

Hung Le, Dung Nguyen, Kien Do, Sunil Gupta, and Svetha Venkatesh. Stable Hadamard Memory: Revitalizing Memory-Augmented Agents for Reinforcement Learning. In *International Conference on Learning Representations*, volume 2025, pp. 73455–73472, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/b6446566965fa38e183650728ab70318-Paper-Conference.pdf.

Jonathan Lee, Annie Xie, Aldo Pacchiano, Yash Chandak, Chelsea Finn, Ofir Nachum, and Emma Brunskill. Supervised Pretraining Can Learn In-Context Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 43057–43083, 2023. doi: 10.52202/075280-1866. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/8644b61a9bc87bf7844750a015feb600-Paper-Conference.pdf.

Chris Lu, Yannick Schroecker, Albert Gu, Emilio Parisotto, Jakob Foerster, Satinder Singh, and Feryal Behbahani. Structured state space models for in-context reinforcement learning. *Advances in Neural Information Processing Systems*, 36:47016–47031, 2023.

Steven Morad, Ryan Kortvelesy, Matteo Bettini, Stephan Liwicki, and Amanda Prorok. POPGym: Benchmarking Partially Observable Reinforcement Learning. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2303.01859.

Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent Model-Free RL Can Be a Strong Baseline for Many POMDPs. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 16691–16723, 2022.

Tianwei Ni, Michel Ma, Benjamin Eysenbach, and Pierre-Luc Bacon. When Do Transformers Shine in RL? Decoupling Memory from Credit Assignment. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2307.03864.

Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International conference on machine learning*, pp. 7487–7498. PMLR, 2020.

Jurgis Pasukonis, Timothy Lillicrap, and Danijar Hafner. Evaluating Long-Term Memory in 3D Mazes. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=yHLvIlE9RGN.

Marco Pleines, Matthias Pallasch, Frank Zimmer, and Mike Preuss. Memory Gym: Towards Endless Tasks to Benchmark Memory Capabilities of Agents. *Journal of Machine Learning Research*, 26 (6):1–40, 2025. URL http://jmlr.org/papers/v26/24-0043.html.

Subhojeet Pramanik, Esraa Elelimy, Marlos C. Machado, and Adam White. AGaLiTe: Approximate Gated Linear Transformers for Online Reinforcement Learning. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=lh6vOAHuvo.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, Chloe Hillier, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1911.05507.

Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables, 2019. URL https://proceedings.mlr.press/v97/rakelly19a.html.

Oleg Shchendrigin, Egor Cherepanov, Alexey K. Kovalev, and Aleksandr I. Panov. Memory Retention Is Not Enough to Master Memory Tasks in Reinforcement Learning. *arXiv preprint arXiv:2601.15086*, 2026. URL https://arxiv.org/abs/2601.15086.

Hado van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Ziyu Wang, Alexander Novikov, Konrad Zolna, Josh S Merel, Jost Tobias Springenberg, Scott E Reed, Bobak Shahriari, Noah Siegel, Caglar Gulcehre, Nicolas Heess, and Nando de Freitas. Critic regularized regression. In *Advances in Neural Information Processing Systems*, volume 33, pp. 7768–7778, 2020.

Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack Rae, Piotr Mirowski, Joel Z. Leibo, Adam Santoro, Mevlana Gemici, Malcolm Reynolds, Tim Harley, Josh Abramson, Shakir Mohamed, Danilo Rezende, David Saxton, Adam Cain, Chloe Hillier, David Silver, Koray Kavukcuoglu, Matt Botvinick, Demis Hassabis, and Timothy Lillicrap. Unsupervised Predictive Memory in a Goal-Directed Agent. *arXiv preprint arXiv:1803.10760*, 2018. URL https://arxiv.org/abs/1803.10760.

P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. doi: 10.1109/5.58337.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient Streaming Language Models with Attention Sinks. In *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/5e5fd18f863cbe6d8ae392a93fd271c9-Paper-Conference.pdf.

Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. VariBAD: A Very Good Method for Bayes-Adaptive Deep RL via Meta-Learning. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1910.08348.

Appendix overview. Appendix A describes the compared memories, their training settings and compute; Appendices B and C give RSM's pseudocode and learning objective; Appendix D proves the memory bounds and accounts for every memory's state; Appendix E specifies the benchmarks, long-running evaluations and statistics; and Appendix F reports additional results.

A Compared Memories and Training

All compared memories receive the same public record: the current observation, previous action and previous reward, encoded by the same stateless timestep encoder with no observation bypass. Each agent carries its own memory state and resets it only at an outer-task boundary. The Transformers have width 256, three layers and eight heads, and the GRU matches their width and depth; RSM uses $C = 32$ and $M = 4$. Within each benchmark, every memory is trained by the same learner (Appendix C) with the same replay, exploration and update schedule (Appendix A.4), in FP32. Table 4 lists the compared memories with their identifiers in our code and their parameter counts, which range over the four benchmarks because their input and action sizes differ; the GRU has about 1.3M fewer parameters than the Transformers.

Table 4: Compared memories, their code identifiers, what each carries forward and its parameter count over the benchmarks it runs on.

Memory	Code identifier	Carried forward	Parameters
RSM (ours)	`raw_summary`	$P_\theta(W)$ at each boundary	3.18–3.39M
Full-history Transformer	`full_context`	Every record's keys and values	3.11–3.32M
GRU	`full_gru`	Recurrent hidden state	1.82–2.03M
Memo	`memo`	Every summary block; jittered segments	3.11–3.32M
Memo, fixed segments	`memo_fixed`	Every summary block	3.11–3.32M
Sliding window	`raw_window`	The last $W = 40$ records per layer	3.12M
RSM w/o memory	`raw_segment`	Z_{init} at each boundary	3.18–3.39M
RSM-R	`raw_summary_residual`	$Z + P_\theta(W)$ at each boundary	3.18–3.39M
RSM detach	`raw_summary_detach`	$P_\theta(W)$, gradient stopped	3.39M

A.1 Memo adaptation

Our comparator follows the author-linked Memo implementation at revision `9e7044f`, adapted to the shared backbone and learner. Each segment produces four summary tokens. Memo samples a segment length from 26–38 once per learner forward and uses 32 during interaction; the fixed-segment variant uses 32 in both settings. Both retain all earlier summary blocks and propagate gradients through them. This comparison matches the present study's conditions rather than reproducing Memo's published task, architecture, and budget in full.

A.2 Sliding window

The sliding window runs the full-history Transformer's backbone under a banded causal mask: every layer attends only to the last $W = 40$ records, through a rolling cache of 40 slots per layer during interaction. $W = 40$ is the largest window whose state fits within RSM's per-agent state (Appendix D.2). Its records keep their absolute positions in the task, so beyond the 500-step training tasks it meets positions it never trained on. It runs on Key-to-Door and MazeRunner with the same learner, 8M-step interaction budget and seeds as every other memory.

A.3 RSM embeddings and controls

Token embeddings. Let p_k be the fixed sinusoidal embedding of slot k, s_j a learned embedding of memory row j, and $\rho^{\mathrm{R}}, \rho^{\mathrm{X}}, \rho^{\mathrm{W}}$ the learned role embeddings. Before embedding dropout, the rows are

$$\begin{aligned} \widetilde{z}_{b,j} &= Z_b[j] + s_j + \rho^{\mathrm{R}} + p_{j-1}, && 1 \le j \le M, \\ \widetilde{e}_{b,i} &= A_\theta(e_{b,i}) + \rho^{\mathrm{X}} + p_{M+i-1}, && 1 \le i \le C, \\ \widetilde{q}_j &= Q^{\mathrm{W}}[j] + s_j + \rho^{\mathrm{W}} + p_{M+C+j-1}, && 1 \le j \le M. \end{aligned} \tag{8}$$

The input projection is A_θ. READ and WRITE rows share memory-slot embeddings but use distinct roles and positions. Both Z_{init} and Q^{W} are learned; the affine map $P_\theta : \mathbb{R}^d \to \mathbb{R}^d$ projects

each final-normalized WRITE output. Parameters are shared across segments, and training retains gradients through the carried summary.

Controls. RSM w/o memory preserves RSM's writer modules and boundary schedule, but discarded writes receive no gradient from later segments. This separately trained control differs from clearing the summary of a frozen RSM policy at evaluation.

A.4 Training settings and compute

Training runs 1,000 epochs; each collects 500 steps from each of 16 agents and then takes 128 gradient updates, for 8M environment steps, reset-only steps included. Saved counters verify this budget. Replay retains complete outer tasks, and before each collection block the agents' states are rebuilt under the current weights (Appendix B). On MazeRunner, following AMAGO's MazeRunner recipe, replay also relabels goals in hindsight for every memory: a replayed task has a random number of its unreached goals replaced by positions the agent reached, and its rewards recomputed; the other benchmarks use no relabeling. Every 50 epochs the greedy policy is scored on the 64 development tasks (Appendix F.7); the reported results use the final checkpoint, with no checkpoint selection. Table 5 lists the settings. Exploration anneals over each agent's first 50,000 steps, as in AMAGO: ϵ-greedy from 1.0 to 0.05; Key-to-Door's bilevel schedule also lowers ϵ within each task, from 1.0 to 0.8 at the start of training and from 0.05 to 0.01 once annealed; and the T-Maze schedule holds ϵ at $0.5/L$ inside the corridor and uses the annealed value at the first decision and at the last decisions before the junction.

Table 5: Training settings, shared by every memory within a benchmark.

	Key-to-Door	CountRecall	MazeRunner	T-Maze
Optimizer	AdamW; learning rate 10^{-4}; weight decay 10^{-3}			
Warm-up; gradient-norm clip	128 updates; 1.0			
Epochs $\times$ steps $\times$ agents	$1{,}000 \times 500 \times 16 = 8$M environment steps			
Updates per epoch; batch	128; 16 complete tasks			
Replay capacity	10,000 complete tasks			
Longest training task (steps)	500	103	500	257
Target-network rate τ	0.003	0.0025	0.003	0.0025
Exploration (ϵ-greedy)	bilevel	standard	standard	T-Maze
Hindsight goal relabeling	no	no	yes	no

Compute. The paper reports 102 fits: seven memories on four benchmarks, the sliding window on two, RSM detach and three further summary lengths, each with three seeds. Every fit trained in FP32 on one GPU at a time, on NVIDIA RTX 3090, RTX A6000, A30, L40S, RTX 6000 Ada, RTX PRO 6000 Blackwell and A100 GPUs, and on RTX 2080 Ti GPUs for the $M = 1$ arm. The records hold at least 880 fit-hours of training time, excluding evaluation. Seventy-two fits were resumed at least once, often on another GPU model, and several fits sometimes shared one GPU. These durations are therefore not hardware-normalized and are not used to compare training efficiency across memories.

B RSM Pseudocode

Algorithm 1 implements cached interaction for one agent; Algorithm 2 gives dense training and agent-state reconstruction. Both encode the public record $u_t = (o_t, a_{t-1}, r_t)$ with the stateless map $e_t = f_\phi(u_t)$. State $s = (Z, \mathcal{K}, \ell, b)$ contains the summary, per-layer key/value caches of $C+2M$ slots, the occupied-slot counter, and the segment counter (the number of completed writes). Append_θ processes embedded rows in causal order (Equation 8), updates the caches and slot count, and returns final-normalized outputs. The boundary operator selects the write rule:

$$\mathrm{Update}_{\omega,\theta}(Z, W) = \begin{cases} P_\theta(W), & \omega = \text{overwrite} \quad \text{(RSM)}, \\ Z + P_\theta(W), & \omega = \text{residual} \quad \text{(RSM-R)}, \\ Z_{\text{init}}, & \omega = \text{no carry} \quad \text{(RSM w/o memory)}. \end{cases} \tag{9}$$

The action is selected from the RECORD representation m; READ and WRITE outputs produce no actions or direct RL losses. A segment that has just received its Cth record keeps its write pending

Algorithm 1 RSM interaction with lazy boundary writes (one agent)

Require: Parameters θ (including ϕ), rule ω, capacities C, M

function RESETAGENT
  return $(Z_{\text{init}}, \text{EMPTYCACHE}(C + 2M), 0, 0)$
end function
function ADVANCE($u, s = (Z, \mathcal{K}, \ell, b)$)
  if $\ell = 0$ **then** ▷ The READ pass is pending
    $(-, \mathcal{K}, \ell) \leftarrow \text{Append}_\theta(\mathcal{K}, \ell, \text{READ}(Z))$
  end if
  if $\ell = M + C$ **then** ▷ Write before admitting the next record
    $(W, \mathcal{K}, \ell) \leftarrow \text{Append}_\theta(\mathcal{K}, \ell, \text{WRITE}(Q^{\text{W}}))$
    $Z \leftarrow \text{Update}_{\omega,\theta}(Z, W)$
    $\mathcal{K} \leftarrow \text{EMPTYCACHE}(C + 2M)$; $\ell \leftarrow 0$; $b \leftarrow b + 1$
    $(-, \mathcal{K}, \ell) \leftarrow \text{Append}_\theta(\mathcal{K}, \ell, \text{READ}(Z))$
  end if
  $e \leftarrow f_\phi(u)$
  $(m, \mathcal{K}, \ell) \leftarrow \text{Append}_\theta(\mathcal{K}, \ell, \text{RECORD}(e, \ell))$
  return $m, (Z, \mathcal{K}, \ell, b)$
end function

until the next record arrives. RESETAGENT is called only at an outer-task reset, so physical-attempt resets preserve the state.

Algorithm 2 Dense RSM unroll and agent-state rebuild

Require: Parameters θ, rule ω

function DENSEUNROLL($U = (u_0, \ldots, u_{N-1}), v$) ▷ v: record validity
  $E \leftarrow f_\phi(U)$; $Z \leftarrow Z_{\text{init}}$; $O \leftarrow [\,]$
  Right-pad E and v to a multiple of C; padded records have $v = 0$
  for each consecutive C-record block (E_b, v_b) **do**
    $X \leftarrow [\text{READ}(Z) \mid \text{RECORD}(E_b) \mid \text{WRITE}(Q^{\text{W}})]$
    $H \leftarrow \text{Norm}_\theta(\text{Transformer}_\theta(X))$ ▷ Causal; padded records masked as keys
    Append the RECORD rows H^{R} to O
    $Z \leftarrow \text{Update}_{\omega,\theta}(Z, H^{\text{W}})$ ▷ Keep the gradient path
  end for
  return $v \odot O_{0:N-1}, Z$
end function

function REBUILD($U^{\text{proc}} = (u_0, \ldots, u_{n-1})$) ▷ No gradients
  $s \leftarrow$ RESETAGENT
  if $n = 0$ **then**
    return s
  end if
  $q \leftarrow \lfloor (n-1)/C \rfloor$ ▷ Segments whose writes have occurred
  if $q > 0$ **then**
    $(-, Z) \leftarrow \text{DENSEUNROLL}(U^{\text{proc}}_{0:qC-1}, \mathbf{1}_{qC})$; $s \leftarrow (Z, \text{EMPTYCACHE}(C + 2M), 0, q)$
  end if
  for $t = qC, \ldots, n-1$ **do**
    $(-, s) \leftarrow \text{ADVANCE}(u_t, s)$ ▷ Re-encode the open segment
  end for
  return s
end function

Attention masks. All layers use ordinary causal content attention. READ rows attend to preceding and self READ rows; RECORD rows attend to the READ rows and to preceding and self RECORD rows; WRITE rows attend to the READ rows, all valid RECORD rows, and preceding and self WRITE rows. C RECORD slots are reserved even for a partial segment, and padded records are masked as attention keys.

Boundaries and deferred writes. A boundary occurs after C admitted records, counting the initial observation and valid reset observations. Cached execution defers the write until the next record arrives: the full segment remains in state, then the model writes, clears the cache, and loads the new READ tokens before admitting that record. Ending an outer task alone triggers no write. Attempt re-

sets preserve memory; an outer-task reset clears the cache and restores Z_{init} independently for each agent. Reset-only steps keep the interface's reward and transition semantics and are not physical actions.

Training and rebuild. Dense training reconstructs complete replayed tasks under current weights and propagates the actor–critic loss (Appendix C) through every carried write. Only valid recorded transitions contribute to the loss; a final write with no following segment has no downstream decision loss. Padding preserves WRITE positions and is masked as attention keys.

Before collection resumes, REBUILD runs without gradients in evaluation mode on U^{proc}, the records consumed since the agent's last outer-task reset. This prefix excludes the pending observation. Using $q = \lfloor (n-1)/C \rfloor$ preserves the deferred write when the last processed segment is full. The resulting state matches cached execution under current weights. Rebuilding avoids replaying stored recurrent states, which become stale as the weights change (Kapturowski et al., 2019). Frozen-policy evaluation uses only Algorithm 1, with no replay or reconstruction.

C OFF-POLICY LEARNING OBJECTIVE

All conditions use AMAGO v3.4.0's scalar actor–critic learner (Grigsby et al., 2024). The actor and four critic heads share the timestep and trajectory encoders. Losses average over valid replay positions and discount heads, with padding masked. Slowly updated target networks are maintained for the actor and critic heads.

Critic regression. For discount γ, let $Q^{\gamma}_{\psi,i}$ denote critic i in scaled reward units and $q^{\gamma}_{\psi,i} = (Q^{\gamma}_{\psi,i} - \mu_\gamma)/\sigma_\gamma$ its PopArt-normalized value (van Hasselt et al., 2016). With reward scale α_r and a pair of target-critic indices $\mathcal{I}_{t,\gamma}$ sampled uniformly with replacement, the one-step target is

$$y^{\gamma}_t = \alpha_r r_{t+1} + \gamma(1-d_{t+1}) \min_{i \in \mathcal{I}_{t,\gamma}} \sum_{a' \in \mathcal{A}} \bar{\pi}^{\gamma}(a' \mid m_{t+1}) \bar{Q}^{\gamma}_{\bar{\psi},i}(m_{t+1}, a'), \tag{10}$$

where bars denote slowly updated target heads, and the critic loss is

$$\mathcal{L}_Q = \mathbb{E}_{\tau,t,\gamma,i}\left[\left(q^{\gamma}_{\psi,i}(m_t,a_t) - \text{sg}\left(\frac{y^{\gamma}_t - \mu_\gamma}{\sigma_\gamma}\right)\right)^2\right]. \tag{11}$$

The operator sg stops gradients. Only the outer-task terminal d stops bootstrapping; an attempt reset preserves future value.

Policy improvement. With the current policy value under critic i and the ensemble-mean advantage of a replay action,

$$\begin{aligned} v^{\gamma}_{i,t} &= \sum_{a \in \mathcal{A}} \pi^{\gamma}_\theta(a \mid m_t) q^{\gamma}_{\psi,i}(\text{sg}(m_t), a), \\ A^{\gamma}_t &= \frac{1}{4}\sum_{i=1}^{4}\left[q^{\gamma}_{\psi,i}(m_t,a_t) - v^{\gamma}_{i,t}\right], \end{aligned} \tag{12}$$

AMAGO combines value maximization with behaviour cloning filtered by a positive advantage (Wang et al., 2020):

$$\mathcal{L}_\pi = -\mathbb{E}_{\tau,t,\gamma}\left[c_Q \min_i v^{\gamma}_{i,t} + c_{\text{BC}}\,\text{sg}(\mathbf{1}\{A^{\gamma}_t > 0\}) \log \pi^{\gamma}_\theta(a_t \mid m_t)\right]. \tag{13}$$

We optimize $\mathcal{L} = \mathcal{L}_\pi + \lambda_Q \mathcal{L}_Q$ with $c_Q = 1$, $c_{\text{BC}} = 0.1$, $\lambda_Q = 10$, and $\alpha_r = 100$. Reward scaling applies to learning targets; policy feedback and evaluation metrics use native, unscaled rewards.

Discount heads. The learner uses seven discount heads, $\Gamma = \{0.1, 0.9, 0.95, 0.97, 0.99, 0.995, 0.999\}$, and the policy acts with the $\gamma = 0.999$ head. Both losses backpropagate through the shared causal encoder over complete outer tasks, including every carried write; READ and WRITE slots have no separate loss.

Gradient through carried summaries. Let $\mathcal{L}_b$ be the part of $\mathcal{L}$ on the decisions of segment b and $\mathcal{L}_{>b} = \sum_{b'>b} \mathcal{L}_{b'}$. Because $\mathcal{L}_{b+1}$ reads Z_{b+1} directly and every later loss reads it only through Z_{b+2}, the gradient that reaches a summary obeys

$$\frac{\partial \mathcal{L}_{>b}}{\partial Z_{b+1}} = \frac{\partial \mathcal{L}_{b+1}}{\partial Z_{b+1}} + \left(\frac{\partial Z_{b+2}}{\partial Z_{b+1}}\right)^{\top} \frac{\partial \mathcal{L}_{>b+1}}{\partial Z_{b+2}}, \tag{14}$$

and trains the writer of segment b through $Z_{b+1} = P_\theta(W_b)$. Under the overwrite, $\partial Z_{b+2}/\partial Z_{b+1} = \partial P_\theta(W_{b+1})/\partial Z_{b+1}$ passes only through attention to the READ rows; under RSM-R it is $I + \partial P_\theta(W_{b+1})/\partial Z_{b+1}$.

D Memory Bounds

D.1 Deployment-state accounting

Proof of Proposition 1. At reset, each agent stores Md summary values and per-layer key/value slabs with $C+2M$ slots. A READ pass uses M slots, followed by at most C RECORD slots. Before the next record, M WRITE queries occupy the reserved final slots and produce the replacement summary. Clearing the cache and loading that summary restores the same layout. Induction over records and boundaries gives $Md + 2L(C + 2M)d$ retained floating-point values, regardless of the number of segments. FP32 storage and the 32-bit and 64-bit counters yield Equation 7, within the supported counter range. Residual and no-carry updates change values but preserve this allocation. □

For $C = 32$, $M = 4$, $d = 256$, and $L = 3$, the per-agent allocation is

$$4(4 \cdot 256) + 4(2 \cdot 3 \cdot 40 \cdot 256) + 12 = 249{,}868 \text{ bytes}. \tag{15}$$

One detached FP32 copy of Z_{init}, $4Md = 4{,}096$ bytes, is shared across the agent batch, so N agents retain $NB_{\text{deploy}} + 4Md$ tensor bytes, excluding parameters, temporary computation, replay, and training-time prefixes and gradients.

D.2 State of the compared memories

Table 6 gives the state that each memory keeps per agent between decisions, counted as for Proposition 1: the history-dependent tensors at FP32 with their counters, excluding parameters and workspace, at their peak within a task of T steps. Table 1 and Figure 3 report these peaks, measured at $T = 4{,}000$. The fixed-size memories keep the same state at every step; the full history and Memo grow with every record or segment.

Table 6: Per-agent state of each memory ($L = 3$, $d = 256$, FP32).

Memory	State kept between decisions	Peak, $T = 4{,}000$
RSM, RSM-R, RSM w/o memory	Summary, buffer caches and counters (Equation 7)	249.9 KB
Sliding window	Keys and values of W records per layer, their positions and a length: $8LWd + 8W + 4$ bytes	246.1 KB
GRU	Hidden state of three layers: $12d$ bytes	3.1 KB
Memo, both variants	Keys and values of four tokens per completed segment and of the open segment	3.24 MB
Full-history Transformer	Keys and values of every record and a length: $8L(T+1)d + 4$ bytes	24.58 MB

D.3 Bounded summary values

Proposition 2 (Bounded summary values)**.** *Fix θ, and let $\nu = \sup_w \|P_\theta(w)\|_2$ over final-normalized rows w, which is finite because the final normalization is a LayerNorm (Ba et al., 2016). For every task, every $b \geq 1$ and every row $j \in \{1, \ldots, M\}$,*

$$\|Z_b[j]\|_2 \leq \nu \quad \textit{(RSM)}, \qquad \|Z_b[j]\|_2 \leq \|Z_{\text{init}}[j]\|_2 + b\,\nu \quad \textit{(RSM-R)}. \tag{16}$$

Under the overwrite, the READ input thus stays in one bounded set however many boundaries a task crosses. The residual rule has only a boundary-dependent bound, which grows with every write. These norm bounds concern the summary's values, not what they encode or how the policy uses them.

Proof of Proposition 2. The final LayerNorm maps a row $x \in \mathbb{R}^d$ to $w = \eta \odot \hat{x} + \zeta$, with gain η, bias ζ and $\hat{x}_k = (x_k - \bar{x})/\sqrt{s^2 + \epsilon}$, where $\bar{x}$ and s^2 are the row's mean and (biased) variance and $\epsilon > 0$. Since $\sum_k (x_k - \bar{x})^2 = ds^2$, we have $\|\hat{x}\|_2^2 = ds^2/(s^2 + \epsilon) < d$, so $\|w\|_2 \le \sqrt{d}\,\|\eta\|_\infty + \|\zeta\|_2$ for every input. Writing $P_\theta(w) = A_{\mathrm{P}} w + c_{\mathrm{P}}$ on each row,

$$\nu \le \|A_{\mathrm{P}}\|_2 \big(\sqrt{d}\,\|\eta\|_\infty + \|\zeta\|_2\big) + \|c_{\mathrm{P}}\|_2 < \infty. \tag{17}$$

Under the overwrite, $Z_b[j] = P_\theta(W_{b-1}[j])$ for $b \ge 1$, so $\|Z_b[j]\|_2 \le \nu$. Under the residual rule, $Z_b = Z_{\mathrm{init}} + \sum_{b'=0}^{b-1} P_\theta(W_{b'})$, and the triangle inequality gives $\|Z_b[j]\|_2 \le \|Z_{\mathrm{init}}[j]\|_2 + b\,\nu$. Both bounds hold for every input sequence, so they apply at every horizon. □

E Evaluation Protocol

Every policy observes the fields listed below together with its previous action and reward; task identities, hidden layouts and evaluator labels never reach it. Training tasks are drawn from identities 0–999,999, development tasks from 1,000,000–1,000,063 and the 256 confirmation tasks from 5,000,000–5,000,255, disjoint from each other. Every reported result is one greedy rollout per confirmation task at the final checkpoint of each run, and nothing is retrained or tuned for the long-running evaluations.

E.1 Benchmarks

Dark Key-to-Door. The start, key and door cells of an 8×8 room are hidden and fixed for the outer task; the agent observes its position, whether it holds the key, and the attempt clock. A task lasts 500 environment steps and holds attempts of at most 50 actions separated by reset-only steps, with memory kept across attempts. The metric is the number of doors completed in 500 steps ($\delta = 1$ door).

CountRecall. A stream of 104 cards from four categories is dealt in a hidden order; the agent observes the dealt and the queried category (one-hot) and a timer, and answers 103 scored queries for the running count of the queried category. The metric is exact accuracy ($\delta = 0.05$). Retention is read by the number of summary rewrites since the queried category was first dealt.

MazeRunner. The walls of a 15×15 maze and the permutation of its five actions are hidden; the agent observes its position, the distances to the four surrounding walls, a timer and the coordinates of three ordered goals, all scaled by the maze size. A task is one episode of at most 500 steps that ends at the third goal or at the timer. The metric is the fraction of goals reached ($\delta = 0.1$; one goal per 500 steps on the long-running tasks).

Passive T-Maze. A cue at the first step fixes the rewarded turn at the junction of a corridor of length L; the agent observes the cue only then; afterwards it observes a flag that marks the junction. Training draws L from the multiples of 32 between 32 and 256 and allows $L + 1$ decisions, with a $-1/128$ penalty for a non-forward action before the junction. The metric is success at $L = 128$ ($\delta = 0.25$).

E.2 Long-running evaluations

Each frozen policy keeps acting in one task for 4,000 steps with its memory carried throughout, and we count its successes in each 500-step window, so a longer task confers no advantage from extra opportunity alone. In the *same task*, the Key-to-Door layout stays fixed while attempts continue, and one MazeRunner map is replayed over repeated attempts: after the third goal or the 500-step timer, the agent restarts on the same maze. A MazeRunner training task holds a single attempt, so on

Table 7: **Paired differences at the training horizon**, RSM minus each memory on the same tasks and seeds: mean, 95% joint seed/task interval, the value on each seed (42 / 100 / 2026) and the verdict of Appendix E.3. T-Maze: every memory reaches 1.000 on every seed (practically equivalent under our rule), except RSM w/o memory ($+0.486$ $[+0.410, +0.536]$; consistent). At the Key-to-Door and CountRecall training horizon, RSM-R differs from RSM by less than δ in mean (Table 1), so neither difference is consistent.

RSM minus	Mean	95% interval	Seeds	Verdict
Key-to-Door, training horizon ($\delta = 1$)				
Full-history Transformer	$+1.6$	$[-1.3, +4.7]$	-0.5 / $+3.8$ / $+1.6$	no consistent difference
GRU	-0.5	$[-5.0, +3.3]$	-4.9 / $+0.4$ / $+2.8$	no consistent difference
Memo	-0.5	$[-2.8, +1.7]$	$+0.5$ / -0.3 / -1.8	no consistent difference
Memo, fixed segments	$+2.3$	$[-1.4, +5.5]$	-1.1 / $+3.9$ / $+4.1$	no consistent difference
Sliding window	$+4.9$	$[+1.4, +9.0]$	$+2.6$ / $+3.1$ / $+9.1$	consistent
RSM w/o memory	$+27.0$	$[+24.4, +29.8]$	$+26.4$ / $+29.2$ / $+25.5$	consistent
CountRecall, training horizon ($\delta = 0.05$)				
Full-history Transformer	-0.009	$[-0.012, -0.007]$	-0.010 / -0.010 / -0.008	equivalent
GRU	$+0.059$	$[+0.017, +0.095]$	$+0.016$ / $+0.065$ / $+0.096$	consistent
Memo	$+0.033$	$[+0.025, +0.042]$	$+0.043$ / $+0.032$ / $+0.024$	equivalent
Memo, fixed segments	$+0.045$	$[+0.037, +0.057]$	$+0.058$ / $+0.038$ / $+0.039$	no consistent difference
RSM w/o memory	$+0.309$	$[+0.292, +0.330]$	$+0.295$ / $+0.303$ / $+0.331$	consistent
RSM detach	$+0.075$	$[+0.062, +0.088]$	$+0.078$ / $+0.088$ / $+0.060$	consistent
MazeRunner, training horizon ($\delta = 0.1$)				
Full-history Transformer	$+0.067$	$[+0.038, +0.098]$	$+0.060$ / $+0.065$ / $+0.077$	equivalent
GRU	$+0.161$	$[+0.123, +0.199]$	$+0.172$ / $+0.169$ / $+0.142$	consistent
Memo	-0.059	$[-0.103, -0.017]$	-0.066 / -0.025 / -0.087	no consistent difference
Memo, fixed segments	-0.033	$[-0.073, +0.005]$	-0.010 / -0.061 / -0.026	equivalent
Sliding window	$+0.053$	$[+0.002, +0.107]$	$+0.010$ / $+0.102$ / $+0.048$	no consistent difference
RSM w/o memory	$+0.066$	$[+0.033, +0.105]$	$+0.049$ / $+0.060$ / $+0.087$	no consistent difference
RSM-R	$+0.048$	$[+0.016, +0.084]$	$+0.061$ / $+0.052$ / $+0.031$	equivalent

MazeRunner every attempt after the first, in both long-running protocols, is new to every memory, whereas Key-to-Door trains with repeated attempts. In a *new task inside the run*, a new Key-to-Door layout is drawn at the first attempt boundary after every 500 steps, and a new MazeRunner maze and goal sequence, with the same action permutation, at the first attempt boundary after every 1,000 steps; the new maps are drawn from identities 6,000,000–6,999,999. Wiping a memory at every attempt boundary isolates what it carries from one attempt to the next.

E.3 Statistics and thresholds

We report means over tasks and seeds. A paired difference between two memories is formed on each task and seed; its 95% interval comes from a joint bootstrap that resamples the three seeds and the tasks with replacement, 2,000 times (Agarwal et al., 2021), and we also report the difference within each seed. CountRecall resamples whole streams, never the queries within one. A difference is *consistent* when its mean exceeds the benchmark's margin δ in magnitude, its joint interval excludes zero and all three seeds agree in sign, *practically equivalent under this rule* when the whole interval lies within $\pm\delta$, and otherwise no consistent difference. These are descriptive reporting labels, not additional hypothesis tests. Tables 7 and 8 list the paired differences between RSM and the other memories behind the comparisons of Section 4, at the training horizon, over long tasks, after task changes and across summary lengths, each with its interval, its value on each seed and its verdict. The margins are practical reporting thresholds rather than significance levels: one event per 500-step window on Key-to-Door and long-running MazeRunner, five accuracy points on CountRecall, ten percentage points of the three-goal fraction on MazeRunner, and 25 percentage points of success on T-Maze. The development thresholds of Figure 13 are 20 doors per task on Key-to-Door, 0.9 exact accuracy on CountRecall, a goal fraction of 0.5 on MazeRunner (descriptive only) and T-Maze success of 1.0 held to the end of training, each on the 64 development tasks. They summarize learning speed and are not used for checkpoint selection.

Table 8: **Paired differences on long-running tasks and summary-length ablations**, reported as in Table 7.

RSM minus	Mean	95% interval	Seeds	Verdict
Key-to-Door, last 500 of 4,000 steps ($\delta = 1$)				
Full-history Transformer	$+17.8$	$[+13.1, +21.5]$	$+20.8$ / $+19.8$ / $+12.8$	consistent
GRU	$+5.4$	$[+1.9, +8.5]$	$+1.6$ / $+6.8$ / $+7.7$	consistent
Memo	$+35.4$	$[+27.0, +42.3]$	$+37.9$ / $+26.0$ / $+42.2$	consistent
Memo, fixed segments	$+14.6$	$[+10.0, +18.8]$	$+17.4$ / $+10.0$ / $+16.4$	consistent
Sliding window	$+41.1$	$[+38.6, +43.8]$	$+42.4$ / $+41.6$ / $+39.3$	consistent
RSM w/o memory	$+43.1$	$[+40.6, +45.6]$	$+42.8$ / $+43.8$ / $+42.7$	consistent
RSM-R	$+0.7$	$[-0.7, +2.1]$	-0.3 / $+0.7$ / $+1.8$	no consistent difference
MazeRunner, last 500 of 4,000 steps ($\delta = 1$)				
Full-history Transformer	$+25.8$	$[+22.5, +29.3]$	$+25.5$ / $+24.8$ / $+27.2$	consistent
GRU	$+7.6$	$[+5.7, +9.5]$	$+7.6$ / $+6.8$ / $+8.2$	consistent
Memo	$+24.8$	$[+21.0, +28.8]$	$+24.8$ / $+22.0$ / $+27.6$	consistent
Memo, fixed segments	$+23.8$	$[+20.1, +27.5]$	$+24.0$ / $+21.6$ / $+25.9$	consistent
Sliding window	$+24.1$	$[+21.2, +27.2]$	$+23.2$ / $+23.9$ / $+25.1$	consistent
RSM w/o memory	$+7.4$	$[+4.6, +10.3]$	$+4.9$ / $+7.7$ / $+9.7$	consistent
RSM-R	$+21.6$	$[+18.4, +25.1]$	$+19.1$ / $+23.7$ / $+22.1$	consistent
Key-to-Door, doors per later layout ($\delta = 1$)				
Full-history Transformer	$+8.6$	$[+6.9, +10.3]$	$+6.9$ / $+10.4$ / $+8.5$	consistent
GRU	$+7.8$	$[+5.6, +9.9]$	$+5.6$ / $+10.0$ / $+7.9$	consistent
Memo	$+8.9$	$[+7.3, +10.5]$	$+7.2$ / $+10.5$ / $+9.2$	consistent
Memo, fixed segments	$+8.8$	$[+7.0, +10.6]$	$+6.9$ / $+10.6$ / $+8.8$	consistent
RSM w/o memory	$+7.5$	$[+5.8, +9.1]$	$+5.8$ / $+9.2$ / $+7.5$	consistent
RSM-R	$+7.6$	$[+5.4, +10.2]$	$+5.4$ / $+10.4$ / $+7.0$	consistent
MazeRunner, goals per 500 steps on the fourth map ($\delta = 1$)				
Full-history Transformer	$+22.6$	$[+19.9, +25.3]$	$+23.2$ / $+21.1$ / $+23.3$	consistent
GRU	$+6.9$	$[+4.9, +9.0]$	$+7.9$ / $+5.7$ / $+7.1$	consistent
Memo	$+21.0$	$[+17.5, +24.4]$	$+21.9$ / $+17.9$ / $+23.3$	consistent
Memo, fixed segments	$+20.0$	$[+17.1, +23.1]$	$+20.9$ / $+17.7$ / $+21.6$	consistent
RSM w/o memory	$+5.1$	$[+2.9, +7.7]$	$+3.6$ / $+4.4$ / $+7.3$	consistent
RSM-R	$+22.6$	$[+19.8, +25.6]$	$+23.6$ / $+20.9$ / $+23.4$	consistent
Key-to-Door, summary length: RSM ($M = 4$) minus M tokens ($\delta = 1$)				
$M = 1$, steps 1–500	-0.5	$[-4.0, +2.7]$	-4.0 / $+2.0$ / $+0.3$	no consistent difference
$M = 1$, steps 3,501–4,000	$+0.7$	$[-1.9, +2.8]$	-1.9 / $+1.5$ / $+2.4$	no consistent difference
$M = 8$, steps 1–500	-1.8	$[-4.7, +1.4]$	-2.5 / $+1.2$ / -4.0	no consistent difference
$M = 8$, steps 3,501–4,000	$+0.7$	$[-1.2, +2.9]$	-0.3 / $+3.1$ / -0.7	no consistent difference
$M = 16$, steps 1–500	-1.0	$[-3.7, +1.7]$	-1.2 / $+1.0$ / -2.9	no consistent difference
$M = 16$, steps 3,501–4,000	$+0.1$	$[-0.8, +1.3]$	$+0.6$ / -0.1 / -0.3	no consistent difference

F Additional Results

Appendices F.1 and F.2 extend Section 4.2; Appendices F.3–F.5 extend Section 4.3; Appendix F.6 extends Section 4.4; and Appendix F.7 shows the training runs of Section 4.1.

F.1 In-context adaptation at the training horizon

Figure 7 shows adaptation within one task at the training horizon. On Key-to-Door every memory except RSM w/o memory raises its door success from about 0.15 on the first attempt to above 0.8 on the ninth by reusing the layout it found, while RSM w/o memory stays near its first-attempt rate. On CountRecall RSM's cumulative accuracy stays close to the full history's through all three rewrites, whereas RSM w/o memory falls once queries reach beyond its buffer and both Memo variants drift down late in the stream. On MazeRunner the fraction of maps reaching each ordered goal is similar across memories, with the GRU lowest. On the T-Maze every memory reaches the goal the cue indicates on every task and on both cue sides, whereas RSM w/o memory turns the same way on every task and succeeds only when the cue points that way.

Table 9: **Memory kept or wiped at every attempt** on one MazeRunner map: goals in the first and last 500 of 4,000 steps (confirmation maps, three seeds).

	Memory kept		Memory wiped	
	1–500	3,501–4,000	1–500	3,501–4,000
RSM (ours)	25.0	26.6	21.4	22.2
Full-history Transformer	8.4	0.8	3.6	0.7
GRU	17.6	19.1	14.9	15.7
Memo	19.5	1.8	21.1	22.2
Memo, fixed segments	18.5	2.8	18.2	18.7
RSM w/o memory	18.4	19.2	17.7	18.2
RSM-R (residual)	20.8	5.0	20.4	21.2

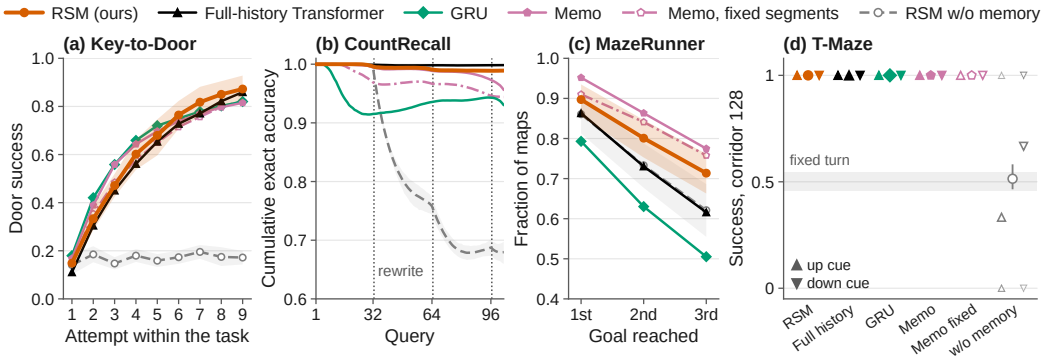


Figure 7: **Within-task adaptation.** (a) Key-to-Door door success by attempt. (b) CountRecall cumulative accuracy; dotted: rewrites after queries 32, 64, 96. (c) Fraction of MazeRunner maps reaching each ordered goal. (d) T-Maze success at corridor 128 over all tasks (95% interval) and by cue side (triangles; faint: seeds); shaded: the success of a fixed turn. Bands in (a–c): RSM and RSM w/o memory.

F.2 Long-running evaluations

Table 9 separates what each memory carries from one attempt to the next on one MazeRunner map (Appendix E.2). The retained-memory mean for RSM exceeds its wiped-memory mean in both the first and last windows, by similar amounts. For both Memo variants, the retained-memory mean falls to a few goals per window, whereas the wiped-memory mean stays near its early value. The full-history Transformer declines even when wiped, so its retained history alone cannot explain the decline, and its retained-versus-wiped difference is negligible in the last window. RSM-R shows the same retained-memory decline (Appendix F.6), while the GRU and RSM w/o memory means change little. Table 9 reports point estimates; we therefore treat these kept-versus-wiped comparisons as descriptive.

Figure 8 resolves the new-task runs of Figure 2g,h in time. With memory never reset, RSM is the only memory whose mean success rate rises within later Key-to-Door layouts: averaged over tasks, its rate within each later layout climbs from under one door per 100 steps to about 3.5, for 9.4 doors per layout against at most 1.9 for every other memory. This average comes from a minority of layouts. RSM completes a door in 34% of later layouts, ranging from 22% to 48% across seeds, and then repeats it at about its adapted rate, but completes none in the rest. This asymmetry suggests that information retained from the preceding layout can bias subsequent search, but the experiment does not identify the content responsible. The other memories that carry the old layout complete a door in at most 15% of later layouts, so in panel (c), where a layout without a door counts as 500 steps, all of them sit near that limit (about 410 steps for RSM, at least 450 for the others). Only RSM w/o memory, which carries no stale layout, finds a door in most later layouts (72%, after about 270 steps), but it cannot repeat what it finds. On MazeRunner, RSM finishes its first attempt on each new map in about 200 steps, as on the first map, against about 300 for the GRU and at least 370 for the growing memories.

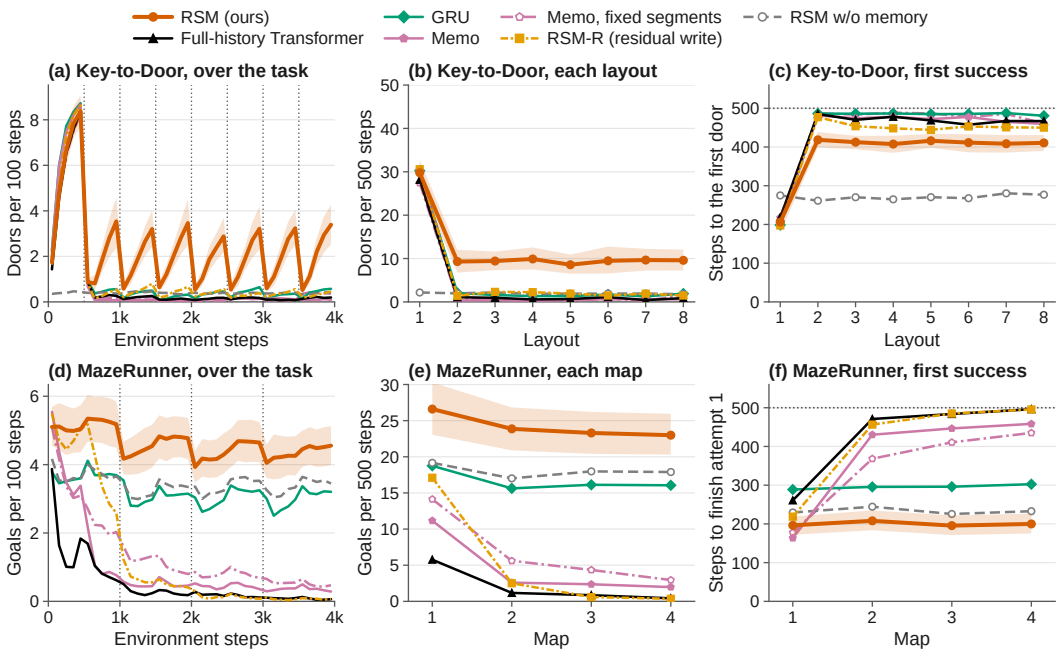


Figure 8: **Re-adaptation without memory resets** (4,000 steps). New Key-to-Door layouts every 500 steps (top); new MazeRunner mazes/goals every 1,000 (bottom). (a,d) Successes per 100 steps; changes dotted. (b,e) Doors per layout/goals per 500 steps. (c,f) Steps to first door (censored at 500)/completed attempt. Band: RSM.

F.3 Summary transplants

At one boundary, we replace every task's written summary with the summary that another task of the same seed read at that boundary, and leave the buffer, the weights and everything else unchanged; the control writes the initial summary at the same boundary (*cleared once*). On Key-to-Door, the first attempt after the boundary visits the donor's key cell far more often than with the task's own summary or with the summary cleared once (Figure 9b), and the misdirected policy completes fewer doors than the cleared one for the rest of the task (Figure 9a): the transplanted summary redirects the policy towards donor-associated cells, and the effect persists across later rewrites. On CountRecall the transplanted policy answers with the counts that the donor's summary implies rather than the true counts until the next rewrite, which moves its answers away from the donor's counts and partly towards the task's own (Figure 9c). Because the early cards are not presented again, subsequent rewards are the only new feedback about whether answers that depend on those early counts are correct.

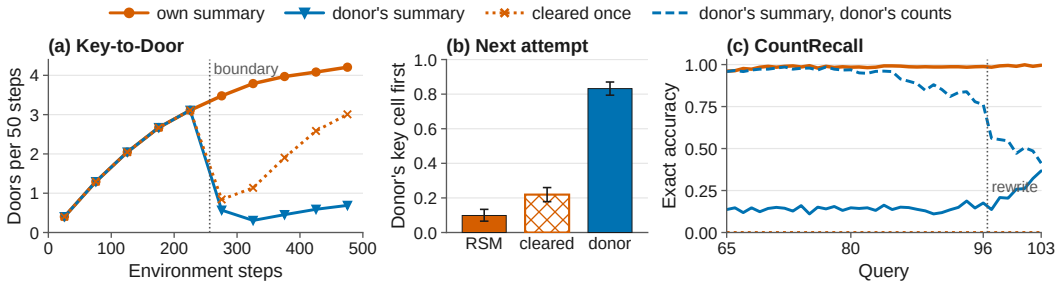


Figure 9: **Summary transplants.** At one boundary, the summary is replaced by a donor task's (blue) or the initial summary (dotted). (a) Key-to-Door doors per 50 steps; transplant before step 257. (b) Fraction of tasks whose next attempt visits the donor's key first. (c) CountRecall accuracy after query 64, scored against true (solid) or donor-consistent (dashed) counts.

F.4 What the summary holds

Figure 10 looks inside the frozen summary (Section 4.3 and Section 4.4). In panels (a–c) and (e–g), one per seed, each dot is one confirmation task, placed by the summary its policy reads at one boundary and coloured by a hidden variable of that task; panels (d,h) report how well a linear read-out recovers each hidden variable from the summary. Ridge regressions fitted on the 64 development tasks and scored on the 256 confirmation tasks recover every count on CountRecall almost perfectly and the row and column of the hidden key on Key-to-Door well, on every seed; the door's position is recovered less well. Projected on its first two principal components, the Key-to-Door summary orders the tasks by where the key lies and the CountRecall summary by the counts, most clearly on seed 42 (the read-outs of the other seeds are as high). Panels (i–k) follow one MazeRunner map over 4,000 steps of repeated attempts: (i,j) compare the summary written at each point of the run with the summary at every other point, and (k) tracks its size. RSM's summaries retain high within-map similarity at a steady norm across rewrites, consistent with a stable map-dependent representation. RSM-R summaries instead drift in one direction until the first and last are anti-correlated, at a norm tens of times larger. Proposition 2 provides a horizon-independent norm bound for the overwrite, but only a boundary-dependent bound for the additive write. These diagnostics are consistent with, but do not establish the cause of, RSM-R's decline on long MazeRunner runs (Section 4.4). A good read-out shows that the information is present in the summary, not that the policy uses it; the interventions of Section 4.3 test its use.

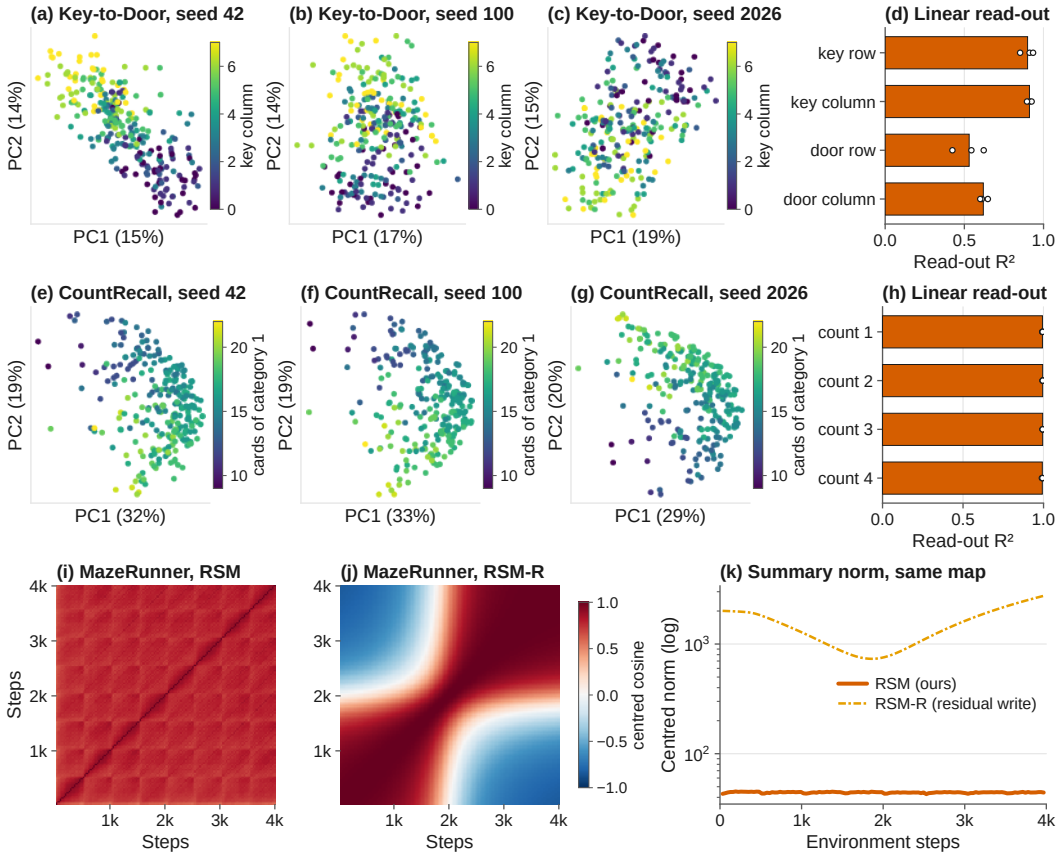


Figure 10: **Summary representations.** (a–c,e–g) First two principal components, fitted per seed on development tasks; colour: key column (Key-to-Door, step 257)/first-category count (CountRecall, query 65). (d,h) Ridge R^2; bars: seed means; dots: seeds. (i,j) Centred cosine similarity of the summaries over repeated MazeRunner attempts on one map; (k) centred norms. Panels (i–k): first 64 confirmation maps.

Probe details. Each read-out is a ridge regression from the flattened summary (Md values) that one segment reads, at step 257 on Key-to-Door (segment 8) and at query 65 on CountRecall (segment 2), to the task's hidden variables: the key's and the door's row and column, or the four running counts. It is fitted per seed on the 64 development tasks and scored as R^2 on the 256 confirmation

tasks. Features and targets are centred by the development means, and the penalty is chosen by five-fold cross-validation on the development tasks from 17 values spaced logarithmically between 10^{-3} and 10^{5}. The principal components are fitted per seed on the same development summaries. For panels (i–k), each seed's flattened summaries are centred by their mean over all maps and segments; panels (i,j) show the cosine similarity between the centred summaries read by two segments of the same map, averaged over maps and then seeds, and panel (k) their mean norm.

F.5 Summary attention

The READ-slot attention share is the attention mass that a decision's heads place on the READ slots, averaged over heads. We compare it with the share $M/(M+i)$ under uniform attention at segment position i and with RSM w/o memory, whose READ slots always hold Z_{init}. Because RSM w/o memory is separately trained, it controls for fixed READ-slot content and position, not for position alone. Figure 11 shows the READ-slot attention share by position in the segment and the effect of biasing decisions' attention away from the summary or the buffer. The decisions assign more attention mass to these slots than RSM w/o memory does, in every layer on Key-to-Door and in the second and third layers on CountRecall, on every seed. Against uniform attention the excess is clear only in the first layer on Key-to-Door and the second on CountRecall; the other layers stay near or below $M/(M+i)$ early in the segment and only slightly above it later. The share hardly depends on where the needed evidence lives: queries whose evidence survives only in the summary, and the first steps of later Key-to-Door attempts, draw only marginally more summary attention than their position predicts. For Figure 6 (Section 4.3), each decision's summary share minus the seed's mean at the same segment position is averaged over the decisions taken at each cell of the Key-to-Door room, on all 256 confirmation tasks at the training horizon, pooling seeds; every cell holds at least 50 decisions.

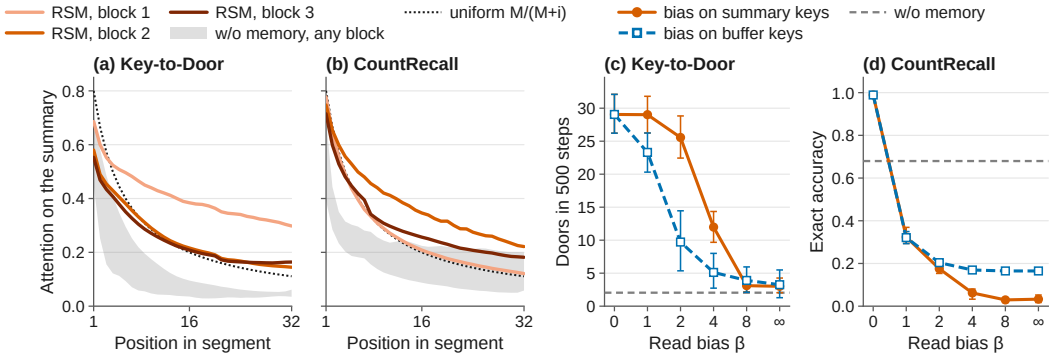


Figure 11: **Summary attention and read interventions.** (a,b) Attention to four summary tokens by segment position/layer; grey band: RSM w/o memory (separately trained fixed-READ baseline); dotted: uniform $M/(M+i)$. (c,d) Performance under attention-score bias $-\beta$ on summary (solid)/earlier buffer keys (dashed); $\beta = \infty$ blocks access; grey dashed: RSM w/o memory.

F.6 Ablations

Figure 12 follows the carry and rewrite ablations over the long-running tasks. On Key-to-Door RSM-R shows no consistent difference from RSM over all 4,000 steps, while RSM w/o memory stays near two doors per window. On one MazeRunner map RSM-R declines from window to window until it falls below RSM w/o memory; the same pattern is absent when its memory is wiped at every attempt (Table 9), implicating the carried state without identifying which property of that state causes the decline. When the maze changes every 1,000 steps, RSM-R falls to a few goals on the second map and almost none after, whereas RSM and RSM w/o memory keep a steady rate.

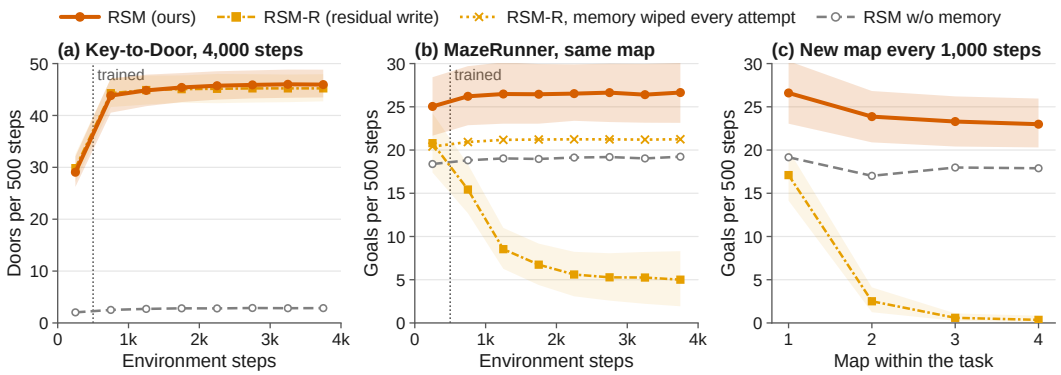


Figure 12: **Carry and rewrite ablations** (Section 4.4; 500-step windows). (a) Key-to-Door doors. (b) MazeRunner goals on one repeated map; crosses: RSM-R with memory wiped each attempt. (c) Goals on four maps, changed every 1,000 steps. Dotted: training horizon.

RSM detach is trained on CountRecall with the same recipe and seeds, its summary detached at every boundary so that no later loss reaches the write. RSM exceeds RSM detach, and RSM detach exceeds RSM w/o memory; both gaps exceed δ on every seed and have joint intervals that exclude zero. The summary-length arms train RSM with $M \in \{1, 8, 16\}$ on Key-to-Door under the same recipe, buffer and seeds, and are compared with $M = 4$ by paired differences on the confirmation tasks at 500 steps and in the last 500 of 4,000 steps with the layout kept (Table 3).

F.7 Learning curves

Figure 13 shows collection return and greedy development performance over the 8M training steps, with every memory trained by the same learner and interaction budget. Every memory reaches the development threshold (Appendix E.3) on every benchmark, except RSM w/o memory on Key-to-Door and CountRecall, and all but RSM w/o memory solve the T-Maze early in training. On Key-to-Door RSM takes longer than every other memory that reaches the threshold, crossing it at 4.4M to 5.2M steps across seeds against at most 3.6M for the others, and then reaches their level by the end of training; on MazeRunner the GRU learns slowest.

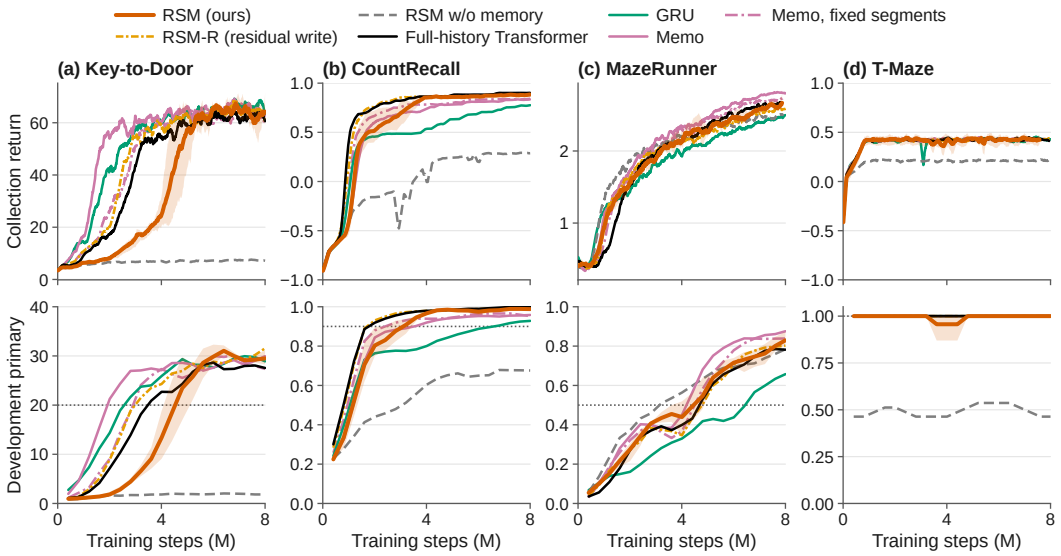


Figure 13: **Learning at matched experience.** Collection return (top)/greedy development performance (bottom) versus training steps for every memory on all four benchmarks. Lines: seed means; band: RSM's seed range; dotted: development thresholds. For display only, collection curves use a centred 25-epoch window and development curves a centred three-evaluation window, both with edge padding; threshold crossings use the raw series.