DESIGNING ASSOCIATIVE WORKING MEMORY FOR PARTIALLY OBSERVABLE REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Partially observable RL creates two complementary memory demands: integrating a trajectory into a control state and preserving specific events for later content-based decisions. We study how the latter demand shapes a working-memory interface. Our task-to-memory formulation derives that interface from the relation a later decision must recover, the retrieval cue, and the event completing the content. For *transition binding*, this process yields Associative Transition Memory (ATM), a slow–fast architecture coupling a recurrent controller to a return-trained associative matrix. ATM treats each completed transition as a memory token: controller context and action form its address, while the resulting observation, outcome, and termination form its content. Two controlled delayed-association tasks test transition-aligned routing efficiency and return-trained selective writing and recall. Explicit transition routing reaches a fixed competence criterion $1.95\times$ sooner in PPO-update count than a parameter-matched implicit interface. At the matched-budget endpoint, ATM achieves higher five-seed rolling-20 training return under both association loads. With four independent associations per episode, it improves over tuned Transition-FWP by 36.2% (0.896 ± 0.013 vs. 0.658 ± 0.046) and leads on all five paired seeds. Separately trained no-read and no-write controls remain near the $1/8$ memoryless reference; phase traces show selective updates. These advantages support the task-to-memory view that RL associative-memory architectures should be specified from task-specific information requirements along five dimensions: memory role, address, content, update, and the learning-and-validation contract.

1 INTRODUCTION

Partially observable control often creates two complementary memory demands. An agent may need to integrate uncertain observations into a trajectory-level control state, yet some decisions depend on a small number of specific events that must remain retrievable by content—for example, the outcome of a previous action. These demands favor different representations. A recurrent state compresses history into a fixed-dimensional control summary (Bakker, 2001; Hausknecht & Stone, 2015). Trajectory attention instead retrieves from a finite memory of stored activations (Parisotto et al., 2020), while associative fast weights preserve content-addressable associations in rapidly updated parameters (Schmidhuber, 1992; Ba et al., 2016; Schlag et al., 2021a). We study how this demand for content-addressable event binding informs working-memory design.

An **information contract** specifies what a later decision must recover, the available cue, when the content becomes complete, its required lifetime, and how memory use is trained and verified. Prior RL fast-memory systems—including FWM, SRWM, and RFWPs—already train the parameters that govern online fast-memory dynamics from RL return (Schlag et al., 2021b; Irie et al., 2022; 2021). Their updates are constructed from per-step activations or recurrent outputs rather than an explicit completed-transition interface. We ask whether making the components of a completed transition explicit in the write interface provides a useful inductive bias.

Associative Transition Memory (ATM) realizes this contract for transition binding (Figure 1). It factorizes a completed transition $(o_t, a_t, z_t, o_{t+1}, d_t)$ into an address from controller context and

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.

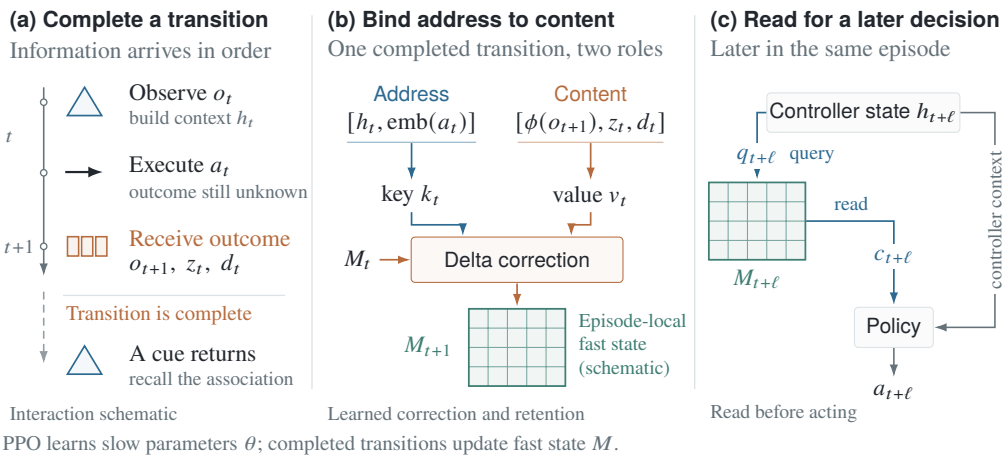


Figure 1: ATM binds completed transitions for later decisions. (a) The write unit becomes complete only after the action's outcome arrives. (b) Controller context and action form the address; the next observation and protocol scalars form the content. Learned correction and retention update the episode-local matrix. (c) A later query retrieves context that joins controller state in the policy, before the next action is chosen. PPO learns slow parameters; completed transitions update the fast state. Symbols and matrix grids are schematic.

action, and content from the subsequent observation, protocol-defined scalar outcome z_t, and termination. A recurrent controller reads the associative matrix before acting; an error-correcting delta rule with learned plasticity and forgetting updates it only after the outcome arrives (§3).

We evaluate the interface prior and the learned write mechanism separately. Contextual Action–Outcome Recall (CAOR) isolates the transition interface through a parameter-matched routing control. Both variants finish at high accuracy, but explicit routing reaches competence $1.95\times$ sooner on the fixed first-hit criterion. Contextual Associative Recall (CAR) then tests return-trained selective writing in a delayed associative-recall task. At the matched-budget endpoint, ATM has the higher five-seed training return under both evaluated association workloads, with the larger advantage when each episode contains four independent associations. Separately trained no-read and no-write controls remain near the memoryless reference, while phase-conditioned traces show that updates concentrate on binding events (§5).

Contributions.

- We formulate a task-to-memory workflow that maps the relation a future decision must recover to five interface choices: memory role, address, content, update, and the learning-and-validation contract (§3).
- We introduce ATM, which realizes transition binding with an action-conditioned address, post-transition outcome content, and an error-correcting delta rule whose learned plasticity and forgetting coefficients are bounded and trained from RL return (§3).
- We evaluate routing sample efficiency with a parameter-matched control, online memory use with separately trained five-seed path controls, and return-trained selective writing with phase-conditioned traces; a task-level query-only diagnostic probes performance without write-phase inputs or reward feedback (§§4–5).

2 Related Work

Recurrent, attention, and episodic memory. DRQN compresses partial-observation history into an LSTM state (Hausknecht & Stone, 2015), while carefully configured recurrent model-free RL can provide a strong baseline across many POMDPs (Ni et al., 2022). GTrXL retrieves from finite Transformer-XL hidden-state memory and uses gates in place of residual connections (Parisotto

et al., 2020); Working Memory Graphs applies multi-head self-attention to a dynamic set of observed and recurrent state vectors (Loynd et al., 2020). Explicit stores preserve item identity: Neural Episodic Control maintains per-action dictionaries of state embeddings and bootstrapped N-step Q estimates (Pritzel et al., 2017); ERLAM builds a graph over embedded states for reverse state–action value propagation (Zhu et al., 2020); and MERLIN passes VAE-derived memory readouts to its action policy while blocking policy gradients into the predictive model (Wayne et al., 2018).

Associative fast-weight primitives. An FWP uses slowly learned parameters to generate context-dependent fast weights that transform subsequent activations (Schmidhuber, 1992). Outer-product FWPs express a write as a key–value association. Linear Transformers accumulate these outer products, DeltaNet replaces accumulation with an error-correcting update and learned step size (Schlag et al., 2021a), and gated variants also learn retention (Yang et al., 2025). These primitives provide complementary choices for association, correction, and forgetting.

Fast weights for adaptive control. FWM's nonlinear LSTM controller generates differentiable delta writes and queries for a separate multilinear associative memory, incorporating its readout into the controller output (Schlag et al., 2021b). RFWPs introduce recurrence in two ways. Delta RNN/LSTM places recurrence inside a delta-updated fast network, whereas Recurrent DeltaNet feeds the previous fast-network output into the projections for the next key, value, query, and learning rate (Irie et al., 2021). SRWM's runtime weight matrix programs its own delta update by producing a modifier key, analyzer query, and learning rate (Irie et al., 2022). In their RL instantiations, return training shapes the projections that govern online fast-memory dynamics from per-step activations or recurrent outputs.

Online neural-memory design frameworks. Titans combines a core, test-time-updated contextual long-term memory, and input-independent persistent memory (Behrouz et al., 2025b). Miras organizes sequence memory by associative memory architecture, attentional bias objective, retention gate, and memory learning algorithm (Behrouz et al., 2026). Nested Learning represents model training as nested optimization problems with distinct context flows and update frequencies (Behrouz et al., 2025a). These frameworks classify online memory by mechanism properties; our task-to-memory formulation instead derives memory role, address, content, update, and validation from the relation a future decision must retrieve.

3 Task-to-Memory Design and ATM

ATM instantiates the task-level information contract as a read–act–write loop (Figure 1). We first specify its transition interface, then describe the controller and memory dynamics, and finally explain how return optimization trains their use.

3.1 Information Contract and Transition Tokens

The information contract begins with the relation a future decision must recover. The surviving cue determines the address, the event boundary fixes content and write timing, and the required lifetime determines retention and reset. The learning signal and evaluation evidence complete the specification. Consider a partially observable interaction with observation o_t, action a_t, PPO reward r_t^{π}, and termination indicator d_t. Let x_{t+1} denote the task-permitted information available after the environment responds. We distinguish the reward used for policy optimization from controller feedback $\bar{r}_t$. A memory-augmented policy can be written as a controller state h_t coupled to a mutable memory state M_t:

$$h_t = F_\theta(h_{t-1}, o_t, a_{t-1}, \bar{r}_{t-1}, d_{t-1}, c_{t-1}), \tag{1}$$

$$q_t = Q_\theta(h_t), \qquad c_t = \mathcal{R}(M_t, q_t), \tag{2}$$

$$a_t \sim \pi_\theta(\cdot \mid h_t, c_t), \tag{3}$$

$$(k_t, v_t, g_t) = \mathcal{W}_\theta(h_t, a_t, x_{t+1}), \tag{4}$$

$$M_{t+1} = \mathcal{U}(M_t, k_t, v_t, g_t). \tag{5}$$

Table 1: Task-to-memory specification for transition binding and its ATM instantiation.

Design axis	Task diagnosis	ATM-derived choice
Memory role	Which information is compressed or addressable?	GRU trajectory context; episode-local matrix associations.
Address	Which cue identifies the event?	Key: controller context and executed action; query: current context.
Content	What is written, and when?	After the transition: next observation, protocol outcome, termination.
Update	How are associations revised and retained?	Delta correction; learned, bounded plasticity and forgetting.
Learning and validation	What trains and verifies memory use?	PPO return; retrained no-read/no-write; query-only; phase traces.

Here $\mathcal{R}$ retrieves from memory, $\mathcal{W}_\theta$ constructs the write variables, and $\mathcal{U}$ updates runtime state. Table 1 derives the ATM design along five axes. ATM uses $x_{t+1} = (o_{t+1}, z_t, d_t)$, where z_t is a protocol-defined scalar outcome channel.

Fast-weight sequence models commonly derive keys, values, queries, and online updates from a per-step activation. For transition binding, the address is available when the agent acts, but the content becomes complete only after the environment returns the outcome. ATM therefore uses the completed transition as its fast-memory token:

$$\tau_t = \big(h_t, a_t, \phi(o_{t+1}), z_t, d_t\big), \tag{6}$$

where h_t summarizes the trajectory prefix ending at o_t. ATM factorizes τ_t into address $[h_t, \mathrm{emb}(a_t)]$ and content $[\phi(o_{t+1}), z_t, d_t]$, while the current context supplies the retrieval query. Earlier per-step activations may encode action and reward implicitly; transition tokenization makes the binding structure explicit at the interface. The read-before-write order keeps the current outcome from affecting a_t while making it available to subsequent decisions.

The underlying store remains an associative fast-weight matrix $M \in \mathbb{R}^{d_k \times d_v}$: accumulation writes $M \leftarrow M + kv^\top$, and query q retrieves $q^\top M$. ATM uses the unnormalized identity-feature numerator of linear attention; the full mechanism also maintains a feature-map normalizer (Katharopoulos et al., 2020). Delta correction and gated decay provide related update primitives (Schlag et al., 2021a; Yang et al., 2025); ATM supplies their inputs through an explicit transition interface.

3.2 Read–Act–Write Dynamics

ATM implements this factorization with a GRU that compresses trajectory context and a matrix that stores context–action–outcome associations. We describe one interaction step in execution order.

Controller and retrieval. The controller is a GRU whose input at step t concatenates the encoded observation, previous action embedding, reward feedback, termination flag, and read context:

$$\begin{aligned} h_t = \mathrm{GRU}\big(h_{t-1},\, [\,&\phi(o_t),\, \mathrm{emb}(a_{t-1}),\, \bar{r}_{t-1},\\ &d_{t-1},\, c_{t-1}\,]\big). \end{aligned} \tag{7}$$

An L2-normalized query retrieves context from the memory,

$$q_t = \mathrm{L2norm}(W_q h_t), \tag{8}$$

$$c_t = \mathrm{LayerNorm}\big(q_t^\top M_t\big), \tag{9}$$

Writing $\pi_t = \pi_\theta(\cdot \mid h_t, c_t)$, the retrieved context directly parameterizes the policy distribution and value estimate:

$$\pi_t = \mathrm{softmax}(W_\pi\, \mathrm{LayerNorm}(h_t + W_a c_t))\,, \tag{10}$$

$$V_t = w_V^\top \mathrm{LayerNorm}(h_t + W_V\, \mathrm{sg}(c_t))\,. \tag{11}$$

The current-step critic branch detaches c_t, while the non-detached c_t enters the controller at step $t+1$. Future value losses can therefore traverse the temporal $c_t \to h_{t+1}$ path under backpropagation through time (BPTT), without a direct current-step critic gradient through the read.

Transition address and content. After sampling a_t from π_t and observing its outcome, ATM constructs the write key from the controller state and an embedding of the executed action:

$$k_t = \mathrm{L2norm}\big(W_k\,[\,h_t,\ \mathrm{emb}(a_t)\,]\big). \tag{12}$$

The value encodes the next observation, the protocol-defined scalar outcome channel, and termination:

$$v_t = \mathrm{MLP}\big(\,[\,\phi(o_{t+1}),\ z_t,\ d_t\,]\big). \tag{13}$$

The content schema is fixed by the interface; the value encoder learns its representation.

Delta update and associative interpretation. An error-correcting delta rule (Schlag et al., 2021a; Irie et al., 2022) revises the addressed value. ATM independently learns correction and global-retention coefficients, the latter related to gated memory decay (Yang et al., 2025). After the outcome arrives, the gate input $g_t = [h_t, \mathrm{emb}(a_t), \phi(o_{t+1}), z_t, d_t]$ concatenates the controller context and completed transition. Two independent scalar affine heads produce the plasticity and forgetting logits:

$$e_t = v_t - k_t^\top M_t, \tag{14}$$

$$s_t^\eta = w_\eta^\top g_t + b_\eta, \qquad s_t^\alpha = w_\alpha^\top g_t + b_\alpha, \tag{15}$$

$$\eta_t = \eta_{\max}\,\sigma(s_t^\eta), \qquad \alpha_t = \alpha_{\max}\,\sigma(s_t^\alpha), \tag{16}$$

$$M_{t+1} = (1-\alpha_t)\,M_t + \eta_t\,(k_t \otimes e_t), \tag{17}$$

We set $\eta_{\max} = \alpha_{\max} = 0.03$; exact initialization appears in the Appendix C. Sigmoid scaling bounds the per-transition coefficients: η_t controls the correction magnitude, while α_t controls retention through $1-\alpha_t$. The head weights and biases belong to θ. Training uses neither gate labels nor an auxiliary gate loss.

The delta term in Equation 17 is one gradient step on the local association objective

$$\ell_t^{\mathrm{assoc}}(M) = \tfrac{1}{2}\left\|k_t^\top M - v_t\right\|_2^2, \tag{18}$$

because $-\nabla_M \ell_t^{\mathrm{assoc}}(M_t) = k_t \otimes e_t$. The complete update can therefore be written as

$$M_{t+1} = (1-\alpha_t)M_t - \eta_t \nabla_M \ell_t^{\mathrm{assoc}}(M_t). \tag{19}$$

The local objective forms episode-specific associations, while return optimization learns their representations, gates, and use by the controller.

For a fixed query q, the correction changes the unnormalized retrieval by $\eta_t(q^\top k_t)e_t$, in addition to global decay. Thus, a write affects queries according to their alignment with its key; the matrix does not allocate an independent slot to each event. Nonorthogonal addresses can interfere, making the learned address geometry relevant to retaining several associations. The actual state change also depends on the residual e_t and existing memory, so plasticity alone does not measure how much an event changes the store.

3.3 Return-Based Learning and State Lifetime

Let θ denote all trainable components: the observation encoder, GRU, key, value, and query projections, write gates, and actor and critic readouts. The fast matrix M_t is episode-local runtime state, updated after each eligible transition by Equation 19; Adam updates θ from experience batches using the PPO objective below. Return gradients train the key, value, and gates through the later consequences of each write, and train the query and policy-facing readout through decisions that use retrieved context. The architecture fixes the candidate content fields, while return optimization learns their representations and continuous update magnitudes.

Each PPO replay chunk starts from stored controller and memory states together with its preceding read context, action, feedback, and termination inputs. Replay then follows the read-before-write order used during acting. The resulting memory dynamics remain differentiable within the chunk, allowing return gradients to reach the representations and gates responsible for later retrievals. Chunk boundaries truncate gradients but preserve M. In the evaluated zero-initialized configuration, episode boundaries reset M to zero and restore the recurrent fields to their start state.

For a trajectory prefix $\mathcal{H}_t$, let b_t be the bootstrap mask, m_t the GAE continuation mask, and $\widetilde{V}_{t+1}$ the appropriate next or terminal-bootstrap value. Generalized advantage estimation gives

$$\delta_t = r_t^{\pi} + \gamma b_t \widetilde{V}_{t+1} - V_{\theta_{\text{old}}}(\mathcal{H}_t), \tag{20}$$

$$\widehat{A}_t = \delta_t + \gamma\lambda m_t \widehat{A}_{t+1}, \qquad \widehat{R}_t = \widehat{A}_t + V_{\theta_{\text{old}}}(\mathcal{H}_t). \tag{21}$$

We use the minibatch-normalized advantage $\overline{A}_t = (\widehat{A}_t - \mu_A)/(\sigma_A + 10^{-8})$. With

$$\rho_t(\theta) = \exp[\log \pi_\theta(a_t \mid \mathcal{H}_t) - \log \pi_{\theta_{\text{old}}}(a_t \mid \mathcal{H}_t)], \tag{22}$$

the clipped policy surrogate and combined PPO loss (Schulman et al., 2017) are

$$\mathcal{L}_\pi = -\mathbb{E}_t\big[\min\big(\rho_t \overline{A}_t, \text{clip}(\rho_t, 1-\epsilon, 1+\epsilon)\overline{A}_t\big)\big], \tag{23}$$

$$\mathcal{L}_V = \tfrac{1}{2}\mathbb{E}_t\left[\left(V_\theta(\mathcal{H}_t) - \widehat{R}_t\right)^2\right], \tag{24}$$

$$\mathcal{L}_{\text{PPO}} = \mathcal{L}_\pi + c_V \mathcal{L}_V - c_H \mathbb{E}_t[\mathcal{H}(\pi_\theta(\cdot \mid \mathcal{H}_t))]. \tag{25}$$

Through the differentiable memory recurrence, this shared objective jointly trains the encoder, controller, memory projections and gates, and actor–critic readouts. Policy gradients traverse the direct read path; value gradients follow the current-step detach and temporal recurrent path defined above.

4 Experimental Setup

CAOR compares two transition-routing interfaces under supplied write timing, while CAR tests return-trained selective writing and a functional read path when every transition is update-eligible. In both tasks, probe/write and distractor steps receive zero reward; each correct query receives $1/Q$, so PPO must credit earlier memory operations through later decisions.

4.1 CAOR: Routing Control

Contextual Action–Outcome Recall (CAOR; Figure 2(a)) isolates how a completed transition is routed into memory. At each active probe, the agent observes a context, acts, and receives a binary outcome that enters z_t but not the observation or immediate reward. A fixed mask marks active probes as complete transitions in both ATM variants. After an information-free distractor, each context cues the outcome of its earlier action. The protocol uses four active contexts and six scored queries; query rewards train PPO and enter controller feedback in both variants. Uniform guessing over the two answer actions yields 0.5 accuracy, versus $1/4$ over the full policy.

The variants share $u = [h, \text{emb}(a), \phi(o'), z, d]$, all trainable modules, gates, read path, parameter shapes, and initialization. Their sole difference is a fixed routing mask: *explicit* routing assigns $[h, \text{emb}(a)]$ to the key and $[\phi(o'), z, d]$ to the value, whereas *implicit* routing sends u to both. Five paired seeds train for 800 PPO updates. We report final query accuracy and, descriptively, the first update reaching 0.95. A separately trained, outcome-blind GRU reference uses four unpaired seeds.

4.2 CAR: Learned Writing and Recall

CAR (Figure 2(b)) evaluates delayed within-episode binding using 32 shuffled association pairs, each spanning successive cue and value timesteps (64 steps total); four or eight are active and the rest are null. After eight distractor steps, 32 shuffled, balanced queries test each cue eight times in the four-association condition and four times in the eight-association condition. The eight-value answer space gives a memoryless reference of $1/8 = 0.125$. Thus, the conditions jointly vary memory load, binding-event density, and query repetition; they are not a pure capacity sweep.

Each episode begins from $M_0 = 0$, so all evaluated fast-memory content must be acquired by within-episode writes and accessed by online reads. Phase and active/null status in the observation identify active cue–value transitions. The environment does not apply these indicators as a write mask: all transitions remain update-eligible, and learned coefficients set update magnitude. Controller reward feedback $\bar{r}_t$ and memory channel z_t are zero; query rewards remain available only to PPO optimization.

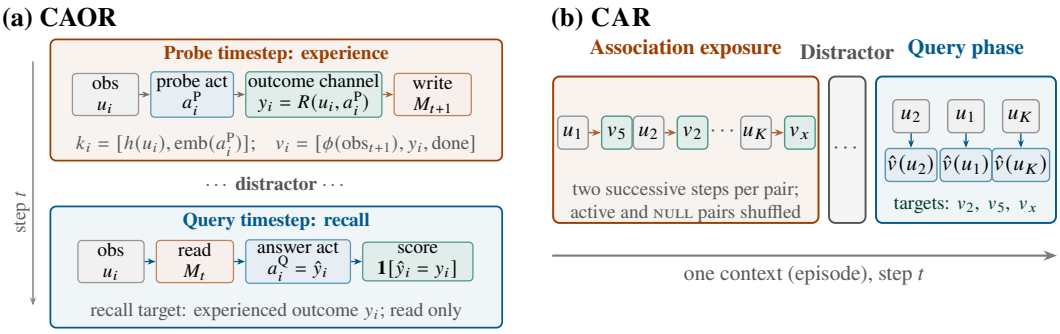


Figure 2: Controlled delayed-association tasks. (a) Contextual Action–Outcome Recall (CAOR): cue u_i and probe action a_i^{P} produce outcome y_i on a dedicated memory channel, which ATM writes and later recalls from u_i. (b) Contextual Associative Recall (CAR) scores cue-only recall after successive-step cue–value exposure and a distractor. CAOR compares routing under supplied write timing; CAR tests return-trained writing with four and eight associations per episode.

Table 2: CAR comparison protocol. ATM and Transition-FWP read before acting and write after each transition; GRU updates recurrently. Every trained configuration receives 2,000 updates (6.656M steps) per seed. Trainable scales are comparable, but Transition-FWP has 253× as many fast-memory elements as ATM.

Model	Parameters	Principal online state per environment
		$h \in \mathbb{R}^{128}$
ATM	145,088	$M \in \mathbb{R}^{16\times16}$
GRU	159,322	$h \in \mathbb{R}^{128}$
Transition-FWP	141,639	$M \in \mathbb{R}^{180\times360}$

Comparisons and controls. For each workload, the five-seed comparison includes zero-initialized ATM, separately trained no-write and no-read variants, GRU, and Transition-FWP. Single-seed GRU *reveal* and ATM *oracle-write* diagnostics test task learnability and the memory path under supplied write selection, respectively. A task-level query-only diagnostic omits write-phase information and reward feedback. Phase-conditioned traces from all five ATM seeds measure update magnitudes by phase.

Transition-FWP adapts the ICML 2021 FWP core (Schlag et al., 2021a) to the continuous observation interface and actor–critic readout, with the same read-before-write timing as ATM. Table 2 records the fixed budget, comparable trainable scale, and unmatched online-state capacity.

Evaluation metrics. Under the fixed budget in Table 2, endpoint training return for each seed averages mean episodic return over updates 1,981–2,000; we then report the mean and sample SD across five seeds. The final checkpoint is evaluated deterministically for 256 episodes per seed.

4.3 Shared Optimization

Across both tasks, we optimize PPO (Schulman et al., 2017) with Adam at learning rate 3×10^{-4}. CAR uses 208-step rollouts with 104-step BPTT chunks; CAOR uses 64-step rollouts with 16-step chunks. The appendix reports the remaining optimization, environment, seed, and artifact details.

5 Results

5.1 Sample Efficiency of Explicit Routing

Explicit ATM reaches the fixed 0.95 criterion sooner in all five paired seeds (Figure 3). With parameters and write timing matched, the 1.95× ratio isolates a sample-efficiency benefit of explicit

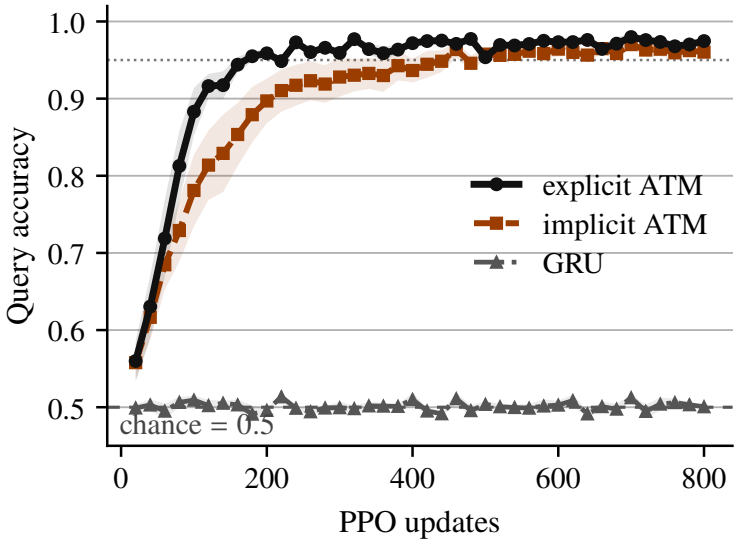


Figure 3: Mean query accuracy during the parameter-matched CAOR routing control. Explicit ATM (solid circles) and implicit ATM (dashed squares) finish at 0.975 and 0.960; explicit first reaches 0.95 in 172 versus 336 updates, a $1.95\times$ ratio in mean first-hit updates. GRU (dash-dotted triangles) remains at the 0.5 reference. Bands show $\pm$SE over five ATM seeds and four unpaired GRU seeds.

key–value routing. Both interfaces finish at high accuracy; the small paired gap ($+0.014$, SE 0.007) confines the conclusion to learning speed rather than final capability. GRU reaches 0.501, matching the intended 0.5 information-limited reference.

5.2 Return-Trained Binding and Selective Writing

CAR next tests return-trained writing when every transition is eligible. At $B = 2{,}000$, ATM achieves higher five-seed endpoint return than tuned Transition-FWP at both workloads, with the larger margin at four associations. Table 3 links this result to held-out recall, retrained path controls, and write selectivity.

ATM leads in endpoint return at both workloads. At the rolling-20 endpoint ending at update $2{,}000$, ATM exceeds tuned Transition-FWP by 0.238 with four associations and 0.045 with eight. ATM is higher on all five paired seeds in each workload; the 95% paired-bootstrap intervals are $[0.194, 0.281]$ and $[0.024, 0.063]$.

With four associations, the endpoint advantage carries over to held-out recall: 0.876 ± 0.024 versus 0.714 ± 0.040. With eight, the held-out means are close at 0.793 ± 0.028 and 0.786 ± 0.055. At four associations, the paired held-out gain is $+0.163$, with ATM higher in all five seeds and a 95% bootstrap interval of $[0.113, 0.210]$. Thus, the endpoint advantage spans both workloads, while the strongest held-out evidence concentrates at four associations.

Online reading and writing support recall. The no-write and no-read variants remain near the $1/8$ memoryless reference. GRU reaches 0.159 and 0.136 held-out accuracy with four and eight associations, respectively, whereas ATM reaches 0.876 and 0.793. The task-level query-only diagnostic also remains at the reference; neither it nor recurrent compression alone approaches ATM recall under this protocol. Each path control is retrained from scratch with one operation disabled, so the evidence supports a functional online read–write path rather than a test-time ablation.

Updates concentrate on binding events. Phase selectivity reaches 3.286 ± 0.133 and 3.198 ± 0.223; the five-seed bootstrap intervals are $[3.184, 3.390]$ and $[3.022, 3.374]$. At both workloads, active cue–value transitions receive larger updates than null, distractor, and query transitions. With

Table 3: CAR results at matched budget $B = 2{,}000$. Bold marks the higher ATM–Transition-FWP mean; five-seed configurations report mean $\pm$ SD. Single-seed GRU reveal and ATM oracle write test task learnability and memory-path feasibility with supplied write selection, respectively; query only is task-level. Held-out accuracy uses 256 episodes per seed; the memoryless reference is 0.125. Phase selectivity is the log ratio of update magnitudes for active bindings versus other transitions; higher values indicate greater selectivity.

	Associations per episode	
Metric / configuration	Four	Eight
Endpoint rolling-20 training return		
ATM	$\mathbf{0.896 \pm 0.013}$	$\mathbf{0.795 \pm 0.027}$
Transition-FWP	0.658 ± 0.046	0.751 ± 0.046
Held-out query accuracy		
GRU reveal	0.999756	0.999756
ATM oracle write	0.912231	0.835571
ATM	$\mathbf{0.876 \pm 0.024}$	$\mathbf{0.793 \pm 0.028}$
Transition-FWP	0.714 ± 0.040	0.786 ± 0.055
No write	0.131 ± 0.016	0.125 ± 0.005
No read	0.126 ± 0.010	0.124 ± 0.004
GRU	0.159 ± 0.032	0.136 ± 0.017
Query only (task)	0.125000	0.124023
Learned write selectivity		
Phase selectivity	3.286 ± 0.133	3.198 ± 0.223

observable phase/status but no environment-applied write mask, larger active-transition updates support return-trained phase-selective writing; all transitions remain eligible, and reward is withheld from recurrent and memory inputs.

6 Discussion, Limitations, and Conclusion

Design implications. Task-to-memory design links a decision's information requirement to the address, content, and lifetime of an association. The two tasks test complementary parts of this link: CAOR isolates the transition-routing prior under supplied write timing, while CAR tests return-trained updates when every transition is eligible. The routing comparison, separately trained read/write controls, and phase traces together support this transition-binding instantiation. They motivate treating the write unit and the information available to each pathway as explicit design choices, alongside the memory's representation and update rule.

This interpretation suggests a division between recurrent context and addressable event information, but the two pathways remain coupled: retrieved content enters the next controller state, and the controller constructs future queries and keys. The experiments therefore establish the usefulness of their combination without showing that either pathway learns a pure or uniquely identifiable representation of belief or events.

Limitations. Tests do not isolate capacity scaling or transfer beyond the fixed cue/value vocabulary and task schema: workloads vary binding density and query repetition, and models differ in online-state capacity. Component attribution needs matched exposure and capacity; broader POMDP and predictive-planning tests remain open.

Conclusion. ATM realizes this interface with an action-conditioned address, post-transition content, and a return-trained associative update. Its controlled-task results connect explicit transition routing to learning efficiency, link online memory use to delayed recall, and show that learned updates concentrate on binding events.

AI USE STATEMENT

The human authors proposed the research ideas and designed the main code architecture. AI assistants supported literature and related open-source code searches, experimental code generation and debugging, test writing and execution, code and data correctness checks, discussions of experimental design, organization of results, manuscript drafting and language editing, and figure and table preparation. Image generation tools were used to explore layouts for the method diagram; the final diagram was drawn as an editable vector graphic. Numerical results come from actual experiment records, and predicted images come from saved model outputs. The human authors led the research decisions and conclusions and take final responsibility for the accuracy and completeness of the paper, code, data, and references.

REFERENCES

Jimmy Ba, Geoffrey E. Hinton, Volodymyr Mnih, Joel Z. Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, pp. 4331–4339. Curran Associates, Inc., 2016.

Bram Bakker. Reinforcement learning with long short-term memory. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 14, pp. 1475–1482. MIT Press, 2001.

Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. Nested learning: The illusion of deep learning architectures. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, pp. 46968–47002. Curran Associates, Inc., 2025a.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, pp. 113506–113543. Curran Associates, Inc., 2025b.

Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. It's all connected: A journey through test-time memorization, attentional bias, retention, and online optimization. In *International Conference on Learning Representations (ICLR)*, 2026.

Matthew Hausknecht and Peter Stone. Deep recurrent Q-learning for partially observable MDPs. In *Sequential Decision Making for Intelligent Agents: Papers from the AAAI Fall Symposium*, number FS-15-06 in AAAI Technical Report, pp. 29–37. AAAI Press, 2015.

Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. Going beyond linear transformers with recurrent fast weight programmers. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 7703–7717. Curran Associates, Inc., 2021.

Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. A modern self-referential weight matrix that learns to modify itself. In *International Conference on Machine Learning (ICML)*, volume 162, pp. 9660–9677. PMLR, 2022.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning (ICML)*, volume 119, pp. 5156–5165. PMLR, 2020.

Ricky Loynd, Roland Fernandez, Asli Celikyilmaz, Adith Swaminathan, and Matthew Hausknecht. Working memory graphs. In *International Conference on Machine Learning (ICML)*, volume 119, pp. 6404–6414. PMLR, 2020.

Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent model-free RL can be a strong baseline for many POMDPs. In *International Conference on Machine Learning (ICML)*, volume 162, pp. 16691–16723. PMLR, 2022.

Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphaël Lopez Kaufman, Aidan Clark, Seb Noury, Matthew Botvinick, Nicolas Heess, and Raia Hadsell. Stabilizing transformers for reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 119, pp. 7487–7498. PMLR, 2020.

Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *International Conference on Machine Learning (ICML)*, volume 70, pp. 2827–2836. PMLR, 2017.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International Conference on Machine Learning (ICML)*, volume 139, pp. 9355–9366. PMLR, 2021a.

Imanol Schlag, Tsendsuren Munkhdalai, and Jürgen Schmidhuber. Learning associative inference using fast weight memory. In *International Conference on Learning Representations (ICLR)*, 2021b.

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwińska, 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, Matthew Botvinick, Demis Hassabis, and Timothy Lillicrap. Unsupervised predictive memory in a goal-directed agent, 2018.

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

Guangxiang Zhu, Zichuan Lin, Guangwen Yang, and Chongjie Zhang. Episodic reinforcement learning with associative memory. In *International Conference on Learning Representations (ICLR)*, 2020.

APPENDIX

A SCOPE

This appendix records implementation, environment, randomization, and artifact details for the experiments in the main paper. It does not add empirical claims or replace material needed to interpret the main results. The accompanying code package provides the implementation, final experiment configurations, anonymized resolved configurations for the admitted runs, aggregate report artifacts, and instructions for rerunning the reported protocols. It does not include pretrained checkpoints or the full raw training and diagnostic records.

B TASKS AND DATA GENERATION

Both evaluation environments generate observations and targets online; no external dataset or preprocessing stage is used. Contextual Action–Outcome Recall (CAOR) uses six probe slots, four active contexts, four distractor steps, and six queries. Contextual Associative Recall (CAR) uses 32 shuffled association pairs, each spanning successive cue and value timesteps (64 exposure steps total), followed by eight distractor steps and 32 queries. The two evaluated CAR workloads expose four or eight cue–value associations per episode. The environment samples the episode instance from its seeded pseudorandom generator.

CAOR uses 16 parallel environments, 64-step rollouts, and 800 PPO updates, for 819,200 environment steps per learned run. CAR uses 16 parallel environments, 208-step rollouts, and 2,000 PPO updates, for 6,656,000 environment steps per learned run. Each final diagnostic evaluation contains 256 episodes and is deterministic with respect to the policy.

C FINAL TRAINING CONFIGURATION

All trained configurations use Adam with learning rate 3×10^{-4}, $\gamma = 0.99$, GAE $\lambda = 0.95$, PPO clip coefficient 0.2, value-loss coefficient 0.5, entropy coefficient 0.001, four replay epochs, advantage normalization, and gradient-norm clipping at 0.5. Each minibatch contains two recurrent chunks, and the value loss is not clipped. CAOR uses 64-step rollouts with 16-step BPTT chunks; CAR uses 208-step rollouts with 104-step chunks. All runs use 16 parallel environments. The reported comparisons use these final settings without per-seed retuning.

The ATM instances use a 128-dimensional GRU, an 8-dimensional action embedding, a 16×16 fast matrix, and $\eta_{\max} = \alpha_{\max} = 0.03$. Both gate biases are initialized to -2. CAOR's explicit and implicit variants share all trainable modules and initialization rules; only the fixed routing mask differs. The CAR no-write and no-read variants retain the ATM architecture and change only the corresponding control mode.

The recurrent baseline uses a 128-dimensional GRU, an 8-dimensional action embedding, and the same flat tanh observation encoder used by the comparison configuration. The Transition-FWP CAR profile uses a 180-dimensional fast weight core, a 128-dimensional input embedding, a 160-dimensional fusion layer, DPFP with feature argument 1, the upstream `ours` update rule, read normalization epsilon 10^{-6}, and write normalization epsilon 0. Its principal online matrix has shape 180×360. Trainable parameter counts are 145,088 for ATM, 159,322 for GRU, and 141,639 for Transition-FWP. Thus, GRU has 9.8% more trainable parameters than ATM, whereas Transition-FWP has only 2.4% fewer.

D RANDOMIZATION AND AGGREGATION

Formal CAR results for ATM, Transition-FWP, GRU, no-read, and no-write use training seeds 0–4. The CAR seed extension repeats initialization, action sampling, and optimization randomness; it is not an independent panel of environment designs. Seed 0 was fixed by the preceding formal gate, and seeds 1–4 were admitted only through the frozen Route-A cell plan. Reveal and oracle-write controls remain seed-0 upper bounds.

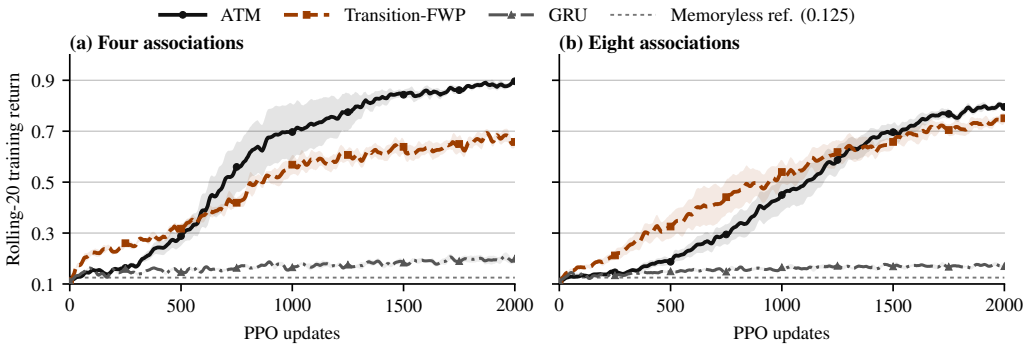


Figure 4: Five-seed CAR learning curves under the matched 2,000-update budget. Each seed is first smoothed with a trailing rolling-20 window; lines and bands show the cross-seed mean and $\pm$SE. Because non-query reward is zero and each correct query contributes $1/32$, episodic return equals the fraction of correctly answered training queries. The dotted line marks the $1/8$ memoryless reference. These curves describe training dynamics; deterministic held-out query accuracy is evaluated only at the final checkpoint.

For CAR, the endpoint training statistic first averages the mean episodic return over updates 1981–2000 within each seed, then reports the mean and sample standard deviation across the five seeds. Held-out query accuracy is computed from 256 evaluation episodes per seed and aggregated in the same order. Paired model differences pair equal training seeds. Selectivity is computed within each episode from median effective-update magnitudes, then aggregated across episodes and seeds. Its confidence interval is a seeded bootstrap over the five seed-level values.

CAOR's endpoint is the final recorded query accuracy at update 800. The sample-efficiency statistic is the first update at which query accuracy reaches 0.95, computed for each of the five paired ATM seeds before averaging. The report uses the fixed plateau window fraction 0.25 and tolerance 0.03.

CAR query-only controls use fixed evaluation seeds 24,000 and 28,000 for the four- and eight-association workloads. Phase traces use 34,000 or 38,000 plus $256s$, where s is the corresponding training seed. Together with the per-run training seeds, these values define the diagnostic evaluation protocol.

E Additional Training Dynamics

Figure 4 shows the trajectories underlying the CAR endpoint statistic: ATM finishes above Transition-FWP under both workloads at 2,000 updates, with the larger separation for four associations.

F Baseline Selection and Comparison Contract

The 2,000-update CAR budget, GRU width, and Transition-FWP profile were fixed before the five-seed extension; seed 0 served as the mechanism gate, and no hyperparameter changed after inspecting seeds 1–4. Transition-FWP adapts the ICML 2021 fast-weight-transformers implementation at commit `ebe13ea` with continuous observation encoding, a learned value projection, read-before-write timing, and actor–critic fusion.

The contract matches updates and environment steps. Trainable scales are close, but ATM's 16×16 matrix has 256 elements versus 64,800 in Transition-FWP's 180×360 matrix. Because the latter remains budget-limited, this is not an asymptotic ranking. The package supplies the final configurations rather than a complete record of exploratory search.

G Computing Environment

The formal CAR run metadata records Ubuntu/Linux kernel 6.8.0-117, glibc 2.35, x86-64, Python 3.12.13, PyTorch 2.11.0 with CUDA 12.8, Gymnasium 0.29.0, NumPy 1.26.4, TensorBoard 2.20.0, and NVIDIA GeForce RTX 4090 GPUs. Historical metadata did not retain the host CPU model, host memory size, GPU driver version, or per-run wall-clock time; no claim is made for those fields.

H Code Package Contents

The package contains the CAOR/CAR implementation, final and anonymized resolved configurations, aggregate reports and curve data, selected analysis scripts, a frozen uv dependency lock, and licenses. The training extra resolves the reported PyTorch 2.11.0 CUDA 12.8 stack. Its registry exposes only CAOR, CAR, and the CAR oracle-write control; reveal uses regular CAR. The README covers installation, smoke tests, protocol reruns, and report locations.

The compact archive omits checkpoints, optimizer states, full logs, per-episode diagnostics, phase traces, and internal ledgers. It supports code inspection, smoke testing, and protocol reruns, but not reevaluation of the original policies or full recomputation of every reported statistic.