ALER: ADAPTIVE LEARNABLE EXPERIENCE REWRITING FOR REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

In partially observable reinforcement learning (RL), a later observation can make stored information obsolete or change what it implies for the next decision. Memory architectures and benchmarks for RL mostly test retention, the ability to keep information unchanged until it is needed. We formalize two further requirements. Rewriting sets the decision-relevant content to a value independent of the old one, and experience fusion transforms the old content by a rule that a later observation specifies. For tasks built from such updates, we count the memory states that a solution needs, and several baselines reach their lowest success rates on compositions that need more states. We introduce ALER (Adaptive Learnable Experience Rewriting), an agent that pairs an LSTM with a slot memory. An independently addressed Gumbel-Softmax write that concentrates its weight on one slot overwrites that slot, and a learned gate fuses the retrieved content with the recurrent state before the policy and value heads. We also introduce RuneMazes, three environments in which rune observations invert, cancel, reset, or repeat updates of a hidden cue under vector and pixel observations. Against seven baselines, ALER reaches a success rate of at least 0.82 in all sixteen Endless T-Maze configurations and at least 0.99 on all five Rune T-Maze compositions, and it has the highest mean success rate on four-branch Rune Multi-Corridor with an `Invert` rune. On pixel-based Rune MiniGrid Memory, it has a higher mean success rate than PPO-LSTM in eight of ten configurations. **Project page: https://anon-aler.github.io/aler/**.

1 INTRODUCTION

An agent in a partially observable environment has to keep information that its current observation no longer shows (Kaelbling et al., 1998). Consider a T-Maze in which the agent sees a cue at the start of a corridor and must turn toward the side it indicates at the junction. If a later corridor shows a new cue, the old cue becomes obsolete and the agent has to rewrite it. If the corridor contains a rune that inverts the cue, the agent has to turn the other way, which it can do only if it still holds the cue and applies the inversion. The same stored cue thus has to be kept, rewritten, or transformed, depending on what the agent observes later.

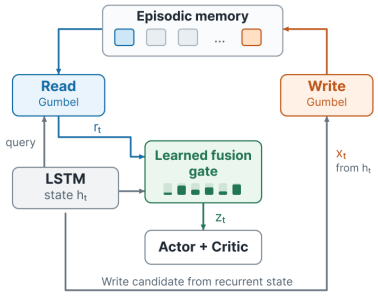

Figure 1: **ALER overview.** The recurrent state h_t queries a slot memory, a learned gate fuses the retrieved vector r_t with h_t into z_t for the actor and critic, and an independently addressed write stores the write candidate x_t.

Memory architectures for RL are mostly built and evaluated for the first of these cases. Recurrent agents (Ni et al., 2022; Cayci & Eryilmaz, 2024) compress the history into one evolving hidden state, and state-space models (Lu et al., 2023) extend the effective horizon with structured recurrent dynamics. Memory-augmented transformers such as GTrXL (Parisotto et al., 2020), HCAM (Lampinen et al., 2021), and DTQN (Esslinger et al., 2022) attend over cached activations of the past, and their gains in RL can come from better memory (Ni et al., 2023). Benchmarks for these agents emphasize retention over long horizons (Wang et al., 2025; Pleines et al., 2025). Endless T-Maze and Color-Cubes (Shchendrigin et al., 2026) add repeated rewriting of stored cues. To our knowledge, no benchmark isolates the transformation of stored content by later observations.

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

We make the three requirements precise in Section 3. An observation calls for retention, rewriting, or experience fusion when it maps the decision-relevant content to itself, to a value independent of it, or to a different function of it. For tasks composed of such updates, we count the memory states that a solution needs. The count separates two solutions. An online solution folds every rune into the stored cue, and a deferred solution keeps the cue unchanged and tracks the runes apart from it. Some compositions force every solution to keep the original cue next to the transformed one.

Adaptive Learnable Experience Rewriting (ALER, Figure 1) is built around this division of work. An LSTM tracks quantities that change at every step. A slot keeps its content while later writes put little total weight on it, and an independently addressed write that concentrates its weight on one slot overwrites that slot in one step. A learned gate combines the retrieved content with the recurrent state.

To test the three requirements, we introduce Rune-Mazes, in which runes invert, cancel, reset, or repeat updates of a cue in vector corridors (Rune T-Maze and Rune Multi-Corridor) and in a pixel-based MiniGrid maze (Rune MiniGrid Memory). Against seven baselines, ALER reaches a success rate of at least 0.82 in every Endless T-Maze configuration and at least 0.99 on every Rune T-Maze composition. It keeps its success up to three times the training corridor length and leads PPO-LSTM in eight of ten Rune MiniGrid Memory configurations. Probes show that ALER stores a new cue with one write and that its policy uses the memory read.

Our contributions are the following.

- A formalization of retention, rewriting, and experience fusion as classes of target updates, with bounds on the memory that a cue-decision task needs (Section 3).
- ALER, which combines an LSTM with an independently addressed slot memory and a learned fusion gate, with a characterization of its write operation (Section 4).
- Rune-Mazes, three environments whose rune compositions have known memory requirements, under vector and pixel observations (Section 5).
- A comparison with seven baselines and an analysis of what ALER stores and how its policy uses it (Sections 7 and 8).

2 Related Work

Recurrent and sequence memory. LSTM policies (Hochreiter & Schmidhuber, 1997; Heess et al., 2015), including PPO-LSTM (Schulman et al., 2017), remain strong POMDP baselines (Ni et al., 2022; Cayci & Eryilmaz, 2024). Recurrent trace units train a recurrent state online by exact real-time recurrent learning in streaming RL (Farr et al., 2026). Transformer-XL (Dai et al., 2019), GTrXL (Parisotto et al., 2020), HCAM (Lampinen et al., 2021), Decision Transformer (Chen et al., 2021), and RATE (Cherepanov et al., 2026c) attend over a context cache, and S5 (Smith et al., 2023; Lu et al., 2023) uses structured state-space recurrence. In these models, stored content changes inside the recurrent update or the attention computation. ALER keeps a recurrent state and adds a slot memory whose content changes only through explicit writes.

External memory and write rules. Neural Turing Machines (Graves et al., 2014) and DNCs (Graves et al., 2016) write to differentiable memory with erase and add operations, and RL-NTM (Zaremba & Sutskever, 2015) trains discrete memory access with reinforcement signals. MERLIN (Wayne et al., 2018), working and episodic memory agents (Fortunato et al., 2019), Neural Episodic Control (Pritzel et al., 2017), and Hopfield-style episodic memory (Chateau-Laurent & Alexandre, 2024) bring external memory into RL. Fast and Forgetful Memory (FFM) (Morad et al., 2023b) and Stable Hadamard Memory (SHM) (Le et al., 2025) apply structured decay and calibration updates to a matrix memory. TARDIS (Gulcehre et al., 2017) also uses near-discrete Gumbel-Softmax addressing, ties its write location to the previously read slot, and is trained with supervised sequence objectives. ELMUR (Cherepanov et al., 2026b) updates layer-local transformer memory offline by imitation for retention tasks. Memo (Gupta et al., 2025) trains RL agents to write summary tokens, and LLM agents learn memory operations from data (Zhang et al., 2025) or receive reminders from a memory agent (Wu et al., 2026). ALER learns separate read and write distributions inside an online PPO agent, so one step can read one slot and overwrite another, and a learned gate controls how the read enters the policy.

Table 1: **Update classes and memory counts of the Rune-Mazes tasks.** Left: effect of each observation on the target y and the rule state, with its class under Definition 1. Right: for each rune composition, the number of rune states Q and the number of target states $Q^\star$ over all rune sequences (Proposition 2), and the exact counts Q_{ep} and $Q^\star_{\text{ep}}$ for sampled episodes (Appendix A). The deferred solution uses KQ states. Rune Multi-Corridor has K branches.

Observation	Effect	
Corridor step, `NoOp`	none	
New cue (Endless)	$y \leftarrow c'$	
`Invert`	$y \leftarrow K{-}1{-}y$	
`SkipNext`	cancels next rune	
Cancelled rune	none	retention
`ResetRules`	$y \leftarrow c$, flags cleared	
`RepeatPrev`	repeats last rune	

Composition	Q	$Q^\star$	Q_{ep}	$Q^\star_{\text{ep}}$
`Invert`	2	2	2	2
`NoOp + Invert`	2	2	2	2
`SkipNext + Invert`	4	4	3	4
`3 Invert + ResetRules`	2	4	2	4
`RepeatPrev + SkipNext + Invert`	10	10	4	6
Multi-Corridor, `Invert`	2	K	2	K

Memory benchmarks. Partially observable versions of MuJoCo (Todorov et al., 2012) and DeepMind Control (Tassa et al., 2018; Ni et al., 2022), POPGym (Morad et al., 2023a), Atari (Bellemare et al., 2013), Obstacle Tower (Juliani et al., 2019), Procgen (Cobbe et al., 2020), POPGym Arcade (Wang et al., 2025), DeepMind Lab (Beattie et al., 2016), Memory Maze (Pasukonis et al., 2022), MiniGrid (Chevalier-Boisvert et al., 2023), Memory Gym (Pleines et al., 2025), MIKASA (Cherepanov et al., 2026a), and image-goal navigation (Mezghani et al., 2022) test memory in RL. Most of their tasks test retention, and some, such as Count Recall in POPGym, require continual updates. Endless T-Maze and Color-Cubes (Shchendrigin et al., 2026) test the rewriting of stored cues. WorldMemArena (Liu et al., 2026) evaluates the writing, maintenance, retrieval, and use of multimodal agent memory. Rune-Mazes add transformations of a stored cue by later runes, and each of its tasks comes with a count of the memory states that a solution needs.

3 Memory Update Requirements

This section assigns every observation in our environments to an update class and counts how much memory a task needs. Section 7 relates these counts to the success of the baselines, and Section 8 compares ALER's representation with the two solutions that they distinguish.

Setup. We consider an episodic POMDP $(\mathcal{S}, \mathcal{A}, \mathcal{O}, P, R, \gamma)$ in which the policy $\pi(a_t \mid h_t)$ conditions on the history $h_t = (o_{1:t}, a_{1:t-1})$. In a *cue-decision task*, the agent receives reward $+1$ at a decision step τ (the junction) if its action matches a target $y_\tau \in \mathcal{Y}$ with $|\mathcal{Y}| = K$, and -1 otherwise. For $t < \tau$, the target y_t is the value that would be correct if the decision followed o_t. It starts at a cue $c = y_1$ shown in the first observation. We assume the following.

(D) The target y_τ is uniform on $\mathcal{Y}$ and independent of the decision-step observation o_τ.

In all our environments, the cue is uniform, independent of the rune placement, and masked after the first step, and the junction observation does not depend on it, so (D) holds.

Update classes. Each observation updates the target, $y_t = F_{o_t}(y_{t-1})$, where the map $F_{o_t} : \mathcal{Y} \to \mathcal{Y}$ may depend on the cue c and on a *rule state* ρ_{t-1}. The rule state holds the task variables that determine how later observations act, such as a pending `SkipNext`, and evolves with the observations alone. In Rune T-Maze, the rule state is the skip flag and the last effective rune, and the invert flag is part of the target, $y = c \oplus \mathit{invert}$.

Definition 1 (Update requirements)**.** Given c and ρ_{t-1}, the observation o_t requires *retention* if F_{o_t} is the identity on $\mathcal{Y}$, *rewriting* if F_{o_t} is constant, and *experience fusion* otherwise.

Rewriting lets an agent discard the old target. Fusion requires the agent to keep the old target and change it. Table 1 (left) classifies the observations of our environments. `SkipNext` cancels the next rune, `RepeatPrev` acts as the last effective rune, and `ResetRules` rewrites the target with the initial cue, so solving a composition that contains it requires retaining the cue.

Memory for one decision. The first result bounds the success of any agent by the number of values its memory can take at the junction.

Proposition 1 (Memory for a decision)**.** *Under (D), let the action at τ depend only on o_τ and a memory variable m_τ with M possible values. Then a correct decision has probability at most* $\min(1, M/K)$.

Each memory value can point to at most one target, and (D) makes every target equally likely. A memoryless policy therefore succeeds with probability at most $1/K$, the **Random** column of Table 4, and a reliable choice among K branches needs at least $\log_2 K$ bits of memory (Appendix A).

Memory for a composed update. A rune composition determines how much memory the target itself requires. For a composition with rune alphabet Σ, let $Q(\Sigma)$ be the number of states of the minimal *rune automaton*, which reads runes from Σ and outputs whether the target is inverted relative to the cue, and call its state the *rune state*. Let $Q^\star(\Sigma)$ be the number of states that the minimal automaton reading the cue and then runes from Σ occupies after the cue, with the target as output. Both counts describe the update rule over all rune sequences in Σ^*. Sampled episodes show each rune candidate at most once, and for them we compute the exact counts $Q_{\text{ep}} \le Q$ and $Q^\star_{\text{ep}} \le Q^\star$ (Appendix A), which order the compositions in the same way.

Proposition 2 (States for a composed update)**.** *(a) Let the corridor hold any rune sequence at any positions, let the agent's state follow $m_t = f(m_{t-1}, o_t)$ and its action be a function of m_t, and let the agent select the correct branch after every cue and every rune sequence in Σ^*. Then at least $Q^\star(\Sigma)$ memory values are reachable at a single corridor position. (b) If the target is a function of the cue and the rune state, then $Q^\star(\Sigma) \le K\,Q(\Sigma)$, and the deferred solution, which stores the cue and the rune state separately, uses exactly $K\,Q(\Sigma)$ states.*

Table 1 (right) lists both counts, which we compute by minimizing the automata defined by the rune semantics (Appendix A). The counts have two uses. First, $Q^\star$ ranges from 2 for `Invert` to 10 for `RepeatPrev + SkipNext + Invert`, and FFM, SHM, and GTrXL reach their lowest success rates on compositions with $Q^\star \ge 4$ (Section 7). Second, the gap between $Q^\star$ and KQ separates two kinds of solutions. An online solution folds each rune into the current target and needs $Q^\star$ states. A deferred solution keeps the cue apart from the rune state and needs KQ states. For `3 Invert + ResetRules`, $Q^\star = 2Q$, because a reset makes the initial cue the target again and every solution has to keep the cue. In Rune Multi-Corridor, `Invert` permutes the K targets, so the online solution needs K states and the deferred solution $2K$.

4 ALER

ALER pairs a recurrent state with a slot memory, following the two-memory view studied in RL (Yang et al., 2023), and assigns the requirements of Section 3 to different components. The LSTM processes every observation and can track the rune state. For retention, a slot keeps its content while later writes put little total weight on it. For rewriting, independently addressed writes overwrite one slot in a single step. For fusion, a learned gate combines the retrieved content with the recurrent state at every step. PPO (Schulman et al., 2017) trains all parts jointly.

One step. At step t (Figure 2), the LSTM receives only the observation features, $(h_t, c_t) = \text{LSTM}(\phi(o_t), h_{t-1}, c_{t-1})$. The recurrent state h_t then queries the memory $M_{t-1} \in \mathbb{R}^{C\times d}$ with slots $m_{t-1,i} \in \mathbb{R}^d$, the gate combines h_t with the retrieved vector r_t into the fused representation z_t, the actor and critic heads both receive z_t, and a write updates the memory to M_t. The retrieved vector enters only the fusion step, so the memory stores projections of past recurrent states and the LSTM computes its update from observations alone. The read uses a query, key, and value projection,

$$\begin{aligned} q^r_t &= \text{normalize}(W_Q h_t), \qquad k^r_{t,i} = \text{normalize}(W_K m_{t-1,i}), \\ w^r_t &= \text{GumbelSoftmax}\big([(q^r_t)^\top k^r_{t,i}]_{i=1}^C, \tau_r\big), \qquad r_t = \sum_{i=1}^{C} w^r_{t,i} W_V m_{t-1,i}. \end{aligned} \tag{1}$$

A learned gate controls the contribution of each retrieved feature,

$$g_t = \sigma(W_g[h_t, r_t] + b_g), \qquad z_t = g_t \odot r_t + (1 - g_t) \odot h_t, \tag{2}$$

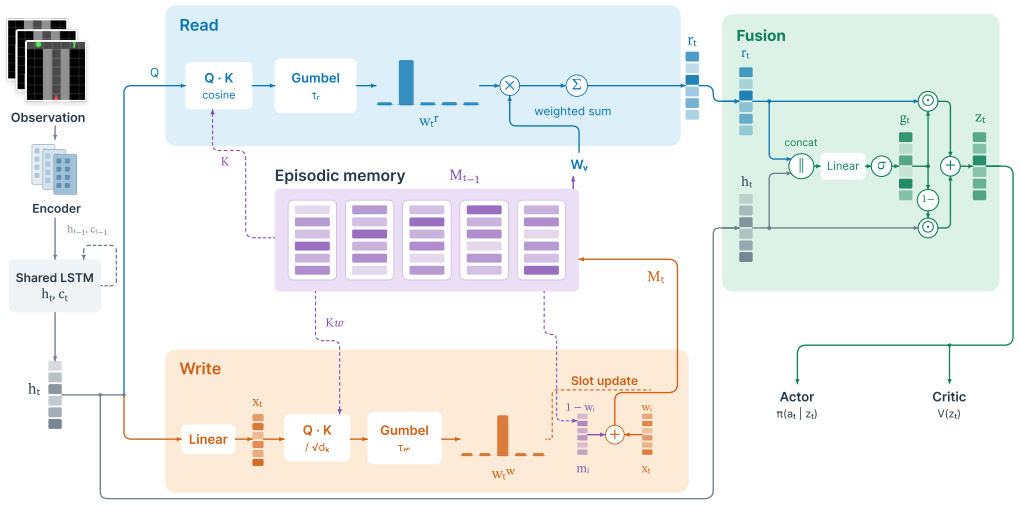

Figure 2: **ALER architecture.** The shared LSTM supplies the read query, the recurrent input to fusion, and the write candidate. Reading uses M_{t-1}, and independent writes produce M_t.

where σ is the sigmoid, $[\cdot,\cdot]$ denotes concatenation, and $\odot$ is elementwise multiplication. The write candidate $x_t = W_{\text{write}} h_t + b_{\text{write}}$ is compared with the slots through separate projections of dimension d_k,

$$\ell^w_{t,i} = \frac{(W_{Q_w} x_t)^\top (W_{K_w} m_{t-1,i})}{\sqrt{d_k}}, \qquad w^w_t = \text{GumbelSoftmax}(\ell^w_t, \tau_w),$$
$$m_{t,i} = (1 - w^w_{t,i}) m_{t-1,i} + w^w_{t,i} x_t. \tag{3}$$

Addressing and initialization. Both addressing steps use $\text{GumbelSoftmax}(\ell, \tau) = \text{softmax}((\ell + \gamma)/\tau)$ with Gumbel noise γ (Jang et al., 2017). The weights stay soft in training and evaluation, gradients flow through the relaxed weights, and fresh noise is drawn at every step, separately for reading and writing. As the temperature decreases, the relaxed samples approach one-hot vectors (Jang et al., 2017). The temperatures are fixed, with $\tau_r = 0.1$ and $\tau_w \in \{0.1, 0.2\}$ in the vector environments. At evaluation, actions are greedy. The memory and the LSTM state are zero at the start of every episode. We use $C = 10$ slots and $d = d_k = 256$ there, and Appendix F lists the MiniGrid values. Appendix B gives the pseudocode and the first step of an episode.

Properties of the write operation. Write weights lie on the simplex, $w^w_t \in \Delta^{C-1}$, in training and evaluation, which gives the write three properties.

Proposition 3 (Soft overwrite). *Let $M_0 = 0$. For every slot i and step $t \geq 1$, (a) $m_{t,i} = \sum_{s \leq t} w^w_{s,i} \prod_{u=s+1}^{t} (1 - w^w_{u,i})\, x_s$, (b) $\|m_{t,i}\| \leq \max_{s \leq t} \|x_s\|$, and (c) if $w^w_{t,i} \geq 1 - \varepsilon$, then $\|m_{t,i} - x_t\| \leq \varepsilon \|m_{t-1,i} - x_t\|$ and $\sum_{j \neq i} w^w_{t,j} \leq \varepsilon$.*

By (a), content written at step s decays only through the write weights that later steps assign to the same slot. By (b), slot norms stay bounded at every horizon, because x_s is an affine map of the bounded LSTM output. By (c), one write with weight close to one overwrites a slot, and every other slot j moves toward x_t by the fraction $w^w_{t,j}$, with $\sum_{j \neq i} w^w_{t,j} \leq \varepsilon$. The constraint $\sum_i w^w_{t,i} = 1$ fixes the total update weight across slots to one, while addressing determines how this weight is distributed. In the trained agent, one write raises the decoding of a new cue from the memory from 0.52 to 0.99 (Section 8), so a single write at a cue step stores the cue.

5 The Rune-Mazes Environments

Rune-Mazes instantiate the update classes of Table 1 in corridors that end in a junction. Together with Endless T-Maze (Shchendrigin et al., 2026), in which each of n consecutive corridors shows a new cue and requires rewriting, they cover all three classes. Figure 3 shows the vector environments.

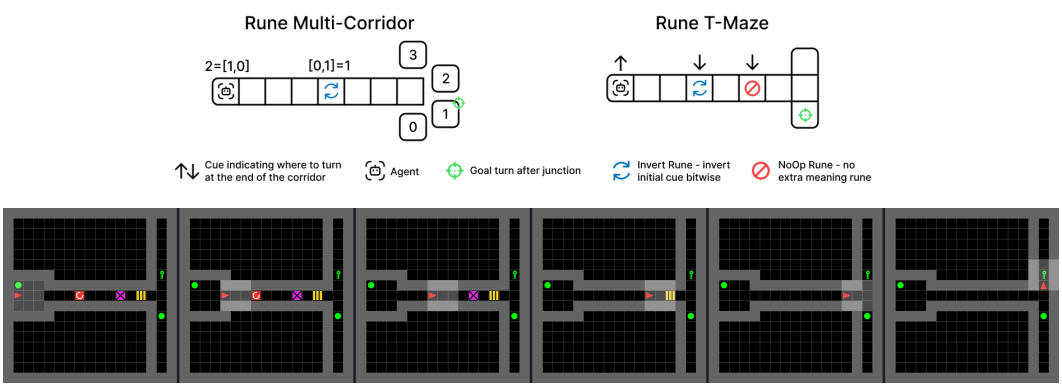

Figure 3: **Rune-Mazes environments. Top left:** Rune Multi-Corridor, in which a multi-bit cue indicates the target branch and an `Invert` rune before the junction changes it. **Top right:** Rune T-Maze, in which the agent must invert the initial cue after an `Invert` rune and ignore a `NoOp` rune. **Bottom:** Rune MiniGrid Memory with `RepeatPrev + SkipNext + Invert`, one episode from the cue (left) to the junction (right). The agent crosses `RepeatPrev` (red), `Invert` (purple), and `SkipNext` (yellow), and each rune fires once on contact (legend in Table 7).

Rune T-Maze. The agent walks a corridor of length 10 and turns left or right at the junction. The first observation shows a binary cue, which is masked afterwards. Each rune of the composition appears at a random corridor position with probability 0.8 and acts on the target as in Table 1. The observation contains a junction indicator, the cue, a direction flag, and the identifier of the rune just crossed. The corridor coordinate is not observed, so the agent has to carry the cue and the rule state to the junction itself. A correct turn gives $+1$, a wrong turn -1, and every other step -0.01. We use the five compositions of Table 1, whose $Q^\star$ ranges from 2 to 10 (Section C.1).

Rune Multi-Corridor. This environment raises the size of the target. The junction has $K \in \{4, 8, 16\}$ branches, and the first observation encodes the target branch as a cue of $\lceil \log_2 K \rceil$ bits, which is masked afterwards. In the `Invert` variant, one rune appears with probability 0.8 and inverts all bits of the target, $y \leftarrow K - 1 - y$. Observations and rewards follow Rune T-Maze (Section C.2).

Rune MiniGrid Memory. This environment tests the same rune semantics under pixel observations. It embeds the runes of Rune T-Maze into MiniGrid Memory (Chevalier-Boisvert et al., 2023), where the agent sees a cue object in a start room and has to reach the matching object at the end of a corridor. Runes are colored floor tiles on corridor cells that every path crosses, placed after the cue room and at least two cells apart. A rune may be visible before contact, fires on entry, and then disappears (Figure 3 and Figure 5). We use the five Rune T-Maze compositions with fixed and random corridor lengths and keep MiniGrid's actions, observations, and rewards (Section C.3).

6 Experimental Setup

Baselines. We compare ALER with seven architectures that cover the main approaches to memory in partially observable RL: PPO-LSTM (Schulman et al., 2017) (recurrent state), GTrXL (Parisotto et al., 2020) (attention over cached activations), Resettable S5 (Lu et al., 2023) (structured state-space recurrence), FFM (Morad et al., 2023b) and SHM (Le et al., 2025) (structured matrix memory), DNC (Graves et al., 2016) in the POPGym implementation (Morad et al., 2023a) (external memory with erase and add writes), and the memory-free PPO-MLP. On Rune MiniGrid Memory, our pixel pipeline trains ALER and PPO-LSTM, the strongest baseline on Rune Multi-Corridor, with the same CNN encoder, LSTM size, and PPO settings (Appendix F).

Training and model selection. For ALER, S5, and DNC on Endless T-Maze and for all architectures on Rune T-Maze and Rune Multi-Corridor, we train five agents per configuration with training seeds 42–46. Rune MiniGrid Memory and the analysis in Section 8 use three training seeds. Every 10^5 environment steps, we evaluate the current checkpoint on 100 validation episodes generated

Table 2: **Endless T-Maze.** SR of the selected checkpoint on the 100 validation episodes, mean ± SEM over five training seeds. A configuration has n corridors of constant length l (Table 8 reports corridor lengths sampled per corridor). n=1 is the classic T-Maze and requires retention only, and every further corridor requires one rewrite. Bold marks every value equal to the best mean in its row.

l	n	PPO-MLP	PPO-LSTM	GTrXL	S5	FFM	SHM	DNC	ALER
5	1	0.50 ± 0.00	$\mathbf{1.00 \pm 0.00}$	0.50 ± 0.19	0.80 ± 0.12	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$
5	3	0.24 ± 0.05	$\mathbf{1.00 \pm 0.00}$	0.17 ± 0.06	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.67 ± 0.24	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$
5	5	0.05 ± 0.02	$\mathbf{1.00 \pm 0.00}$	0.04 ± 0.02	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.84 ± 0.13	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$
5	10	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$	0.61 ± 0.17	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.23 ± 0.22	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$
10	1	0.50 ± 0.00	$\mathbf{1.00 \pm 0.00}$	0.43 ± 0.01	0.80 ± 0.12	$\mathbf{1.00 \pm 0.00}$	0.84 ± 0.16	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$
10	3	0.16 ± 0.02	$\mathbf{1.00 \pm 0.00}$	0.17 ± 0.01	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.93 ± 0.07	0.00 ± 0.00	0.82 ± 0.17
10	5	0.04 ± 0.00	0.69 ± 0.31	0.19 ± 0.12	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.52 ± 0.24	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$
10	10	0.00 ± 0.00	0.83 ± 0.17	0.02 ± 0.02	$\mathbf{1.00 \pm 0.00}$	0.73 ± 0.27	0.02 ± 0.01	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$

with seed 123 and select the checkpoint with the highest success rate, breaking ties by mean return. All result tables report the success rate of this *selected checkpoint* as mean ± standard error (SEM) over training seeds. The selection rule is the same for all eight architectures. The analysis in Section 8 uses episodes independent of checkpoint selection: evaluation seeds 1–3 with 100 episodes each for gate experiments, and 128 further episodes per checkpoint with an episode-disjoint $75/25$ split for probes.

Hyperparameters and capacity. On Endless T-Maze, PPO-LSTM, GTrXL, SHM, FFM, and PPO-MLP use the architecture-specific configurations published by Shchendrigin et al. (2026). For ALER and S5 on Endless T-Maze and for all architectures on Rune T-Maze and Rune Multi-Corridor, we first sweep architecture parameters and then PPO parameters with two million environment steps per candidate. We keep one hyperparameter setting per model and environment family, selected on representative configurations of that family, and use it for every rune composition and branch count (Appendix F). On the vector tasks, ALER has about 1.1M parameters, about 0.5M of them in the memory module, and PPO-LSTM about 2.3M.

Research questions. Each rune composition and branch count is a separate training task, so the experiments measure whether an agent learns a composed update rule from reward. **RQ1** (rewriting): does ALER replace obsolete cues in Endless T-Maze, at the training horizon and beyond it? **RQ2** (fusion): does ALER learn composed updates of a binary cue in Rune T-Maze and of a cue with $K \in \{4, 8, 16\}$ values in Rune Multi-Corridor? **RQ3** (fusion from pixels): does ALER learn the same rune compositions from pixel observations in Rune MiniGrid Memory? **RQ4** (mechanism): where does ALER keep the cue and the rule state, and how does its policy use the memory read? Section 7 answers RQ1–RQ3, and Section 8 answers RQ4.

7 Results

RQ1: rewriting in Endless T-Maze. ALER replaces obsolete cues reliably at the training horizon. It reaches a success rate of 1.00 in fifteen of the sixteen Endless T-Maze configurations and 0.82 ± 0.17 in the remaining one (Table 2 for constant corridor lengths and Table 8 in Section D.1 for random ones). ALER and S5 are the only architectures with a success rate of at least 0.80 in every configuration. PPO-LSTM reaches 1.00 in thirteen configurations and 0.69 ± 0.31 to 0.98 ± 0.02 in the other three. The success of GTrXL, SHM, FFM, and PPO-MLP generally decreases as the number of corridors, and with it the number of rewrites, grows.

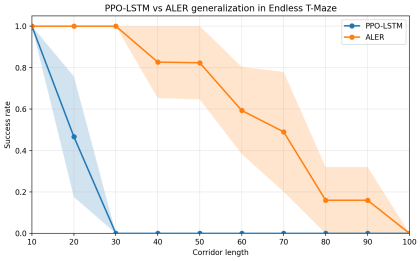

Figure 4: **Endless T-Maze horizon extrapolation.** Mean success rate over five training seeds for agents trained on 10 corridors of length 10 and evaluated on 10 corridors of lengths 10 to 100. Shaded regions show the SEM.

ALER also keeps its success on corridors longer than those seen in training. Both agents in Figure 4 are trained on 10 corridors of length 10 and evaluated on 10 corridors of lengths 10 to 100, which changes the horizon and keeps the task semantics. ALER stays at a success rate of 1.00 up to length 30, three times the training length, and PPO-LSTM falls below 0.5 at length 20 and reaches zero at length 30. Proposition 3(a) is consistent with this gap,

Table 3: **Rune T-Maze.** Success rate of the selected checkpoint on the 100 validation episodes, mean ± SEM over five training seeds. Compositions are ordered by $Q^\star$ and $Q^\star_{\text{ep}}$, which give the same order (Table 1). Bold marks every value equal to the best mean in its row.

Composition	$Q^\star$	PPO-MLP	PPO-LSTM	GTrXL	S5	FFM	SHM	DNC	ALER
`Invert`	2	0.42 ± 0.11	0.84 ± 0.04	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.72 ± 0.09	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$
`NoOp + Invert`	2	0.51 ± 0.00	0.78 ± 0.00	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.74 ± 0.06	0.48 ± 0.07	$\mathbf{1.00 \pm 0.00}$
`SkipNext + Invert`	4	0.51 ± 0.01	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.65 ± 0.08	0.53 ± 0.00	0.56 ± 0.02	$\mathbf{1.00 \pm 0.00}$
`3 Invert + ResetRules`	4	0.42 ± 0.11	0.81 ± 0.05	$\mathbf{0.99 \pm 0.00}$	$\mathbf{0.99 \pm 0.00}$	0.59 ± 0.03	0.51 ± 0.02	0.48 ± 0.11	$\mathbf{0.99 \pm 0.00}$
`RepeatPrev + SkipNext + Invert`	10	0.41 ± 0.10	0.83 ± 0.02	0.92 ± 0.08	$\mathbf{0.99 \pm 0.01}$	0.62 ± 0.01	0.55 ± 0.01	0.50 ± 0.00	$\mathbf{0.99 \pm 0.01}$

Table 4: **Rune Multi-Corridor.** SR of the selected checkpoint on the 100 validation episodes, mean ± SEM over five training seeds. K branches correspond to cues of 2, 3, and 4 bits. The no-rune variant requires retention of the cue, and the `Invert` variant requires fusion. **Random** is the success rate of a uniform guess, $1/K$. Bold marks every value equal to the best mean in its row.

Setting	K	PPO-MLP	PPO-LSTM	GTrXL	S5	FFM	SHM	DNC	ALER	Random
No runes	4	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$	0.00 ± 0.00	0.55 ± 0.14	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$	0.25
	8	$\mathbf{0.03 \pm 0.03}$	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	$\mathbf{0.03 \pm 0.03}$	0.00 ± 0.00	0.00 ± 0.00	0.125
	16	0.02 ± 0.02	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.01	0.00 ± 0.00	0.08 ± 0.01	$\mathbf{0.10 \pm 0.03}$	0.0625
`Invert`	4	0.05 ± 0.05	0.77 ± 0.00	0.00 ± 0.00	0.55 ± 0.16	0.00 ± 0.00	0.00 ± 0.00	0.21 ± 0.02	$\mathbf{0.83 \pm 0.17}$	0.25
	8	0.02 ± 0.02	$\mathbf{0.35 \pm 0.10}$	0.00 ± 0.00	0.12 ± 0.06	0.03 ± 0.03	0.06 ± 0.04	0.16 ± 0.04	0.00 ± 0.00	0.125
	16	0.00 ± 0.00	$\mathbf{0.10 \pm 0.03}$	0.00 ± 0.00	0.07 ± 0.04	0.02 ± 0.01	0.00 ± 0.00	0.05 ± 0.02	0.00 ± 0.00	0.0625

since an earlier write in a slot decays by $\prod_u (1 - w^w_{u,i}) \geq 1 - \sum_u w^w_{u,i}$, a bound set by the total later write weight on the slot at any corridor length.

RQ2: fusion in Rune T-Maze. ALER learns all five composed updates. It reaches 1.00 on `Invert`, `NoOp + Invert`, and `SkipNext + Invert` and 0.99 on `3 Invert + ResetRules` and `RepeatPrev + SkipNext + Invert` (Table 3). On 300 episodes from evaluation seeds 1–3, unused in checkpoint selection, the ALER agents of Section 8 reach 1.00 ± 0.00 on `Invert` and `RepeatPrev + SkipNext + Invert` (Table 10). ALER and S5 are the only architectures with a success rate of at least 0.99 on all five compositions. GTrXL reaches at least 0.99 on four compositions and 0.92 ± 0.08 on `RepeatPrev + SkipNext + Invert`, the composition with the largest $Q^\star$. For FFM and SHM, success follows the counts of Proposition 2. Each has a higher success rate on both compositions with $Q^\star_{\text{ep}} = 2$ than on any composition with $Q^\star_{\text{ep}} \geq 4$. PPO-MLP stays between 0.41 and 0.51, close to the bound of 0.5 that Proposition 1 gives for a memoryless policy.

RQ2: fusion in Rune Multi-Corridor. At $K = 4$ without runes, ALER and PPO-LSTM share the highest success rate, 1.00, and S5 reaches 0.55 ± 0.14 (Table 4). With one `Invert` rune, ALER has the highest mean, 0.83 ± 0.17, against 0.77 ± 0.00 for PPO-LSTM and 0.55 ± 0.16 for S5. At $K \geq 8$, where the cue needs $\log_2 K$ bits (Proposition 1), no architecture exceeds 0.35. There, a truncated episode returns -0.20 and a uniform guess $2/K - 1$ before step penalties, -0.75 at $K = 8$ and -0.875 at $K = 16$ (Section C.2), so an agent that has not decoded the cue earns more by never selecting a branch. Across the three vector environments, ALER is the only architecture that reaches at least 0.82 in every Endless T-Maze configuration, at least 0.99 on every Rune T-Maze composition, and the highest mean at $K = 4$ with and without the `Invert` rune.

Table 5: **Rune MiniGrid Memory.** Success rate of the selected checkpoint within 30M steps on the 100 validation episodes, mean ± STD over three training seeds. Bold marks the higher mean. Rows are compositions with `I` = `Invert`, `N` = `NoOp`, `S` = `SkipNext`, `R` = `ResetRules`, and `P` = `RepeatPrev`.

	Fixed length		Random length	
	PPO-LSTM	ALER	PPO-LSTM	ALER
`I`	$\mathbf{0.80 \pm 0.03}$	0.69 ± 0.09	0.93 ± 0.07	$\mathbf{0.94 \pm 0.06}$
`N+I`	$\mathbf{0.94 \pm 0.06}$	0.77 ± 0.03	0.67 ± 0.07	$\mathbf{0.92 \pm 0.08}$
`S+I`	0.51 ± 0.03	$\mathbf{0.59 \pm 0.01}$	0.62 ± 0.07	$\mathbf{0.63 \pm 0.05}$
`3I+R`	0.53 ± 0.04	$\mathbf{0.57 \pm 0.01}$	0.53 ± 0.01	$\mathbf{0.57 \pm 0.01}$
`P+S+I`	0.49 ± 0.03	$\mathbf{0.60 \pm 0.00}$	0.56 ± 0.00	$\mathbf{0.69 \pm 0.06}$

RQ3: fusion from pixels in Rune MiniGrid Memory. From pixel observations on Rune MiniGrid Memory, ALER has a higher mean success rate than PPO-LSTM in eight of ten configurations and in all five random-length variants, 0.70 against 0.66 on average (Table 5). Its largest lead is on random-length `NoOp + Invert`, 0.92 ± 0.08 against 0.67 ± 0.07. On fixed-length corridors, ALER leads on the three compositions with $Q^\star \geq 4$.

8 What ALER Stores and How It Uses Memory

Linear probes, scored by balanced accuracy on held-out episodes, and evaluation-time experiments on the gate test three mechanisms from Propositions 2 and 3: one write stores a new cue, stored content persists, and a deferred solution keeps the cue apart from the rune state (Appendix E).

The policy needs both the recurrent state and the read. We ablate the read path and the learned fusion at evaluation. On Endless T-Maze and on Rune T-Maze `Invert` and `RepeatPrev + SkipNext + Invert`, we replace the learned gate by a constant g_t, with frozen weights and active memory writes. The constant $g_t = 0$ removes the read path and passes h_t alone, $g_t = 0.5$ replaces learned fusion by a fixed equal mixture, and $g_t = 1$ passes r_t alone. Learned fusion has the highest mean success rate on all three tasks (Table 10), and a constant $g_t = 1$ gives at most 0.26. Setting $g_t = 0$ lowers success by 0.24 on Endless T-Maze with 20 corridors of length 50, beyond the training range, and by 0.17 and 0.28 on Rune T-Maze `Invert` and `RepeatPrev + SkipNext + Invert`.

One write stores a new cue, and the cue stays decodable. In Endless T-Maze at length 50, the first observation of a new cue raises its decoding from 0.52 ± 0.01 in the memory before the write, M_t^-, to 0.99 ± 0.01 after it, M_t^+, since by Proposition 3(a) the write adds $w_{t,i}^w x_t$ to every slot and x_t carries the cue. Along the corridor, M_t^+ decodes the cue with 0.94 to 0.99 at all five probed positions, and h_t, x_t, and z_t with 1.00. Appendix E adds previous-cue and addressing probes.

The initial cue is decodable from the slot memory, and the rune flags from the recurrent state. On Rune T-Maze `RepeatPrev + SkipNext + Invert`, the decoding pattern of ALER matches the division of the deferred solution of Proposition 2 (Figure 11). The memory M_t^+ decodes the initial cue with 0.85 ± 0.06 and the `Invert` and `SkipNext` flags with 0.59 ± 0.01 and 0.47 ± 0.09, where chance is 0.5. The recurrent state h_t decodes the two flags with 0.70 ± 0.02 and 0.64 ± 0.03 and the initial cue with 0.98 ± 0.01, so the cue is available on both paths.

The gate reweights features at rule observations. If the gate carries out fusion, it should change most at rune steps. At `Invert` rune steps in Rune T-Maze, the mean gate coefficient decreases by 0.03 ± 0.01. Compared with non-rune steps matched by position, direction, and initial cue, the mean absolute change per component, D_t, is higher by 0.02 to 0.04 at every rune type of `Invert` and `RepeatPrev + SkipNext + Invert`, and each difference is positive in every training seed (Section E.3).

9 Conclusion

We proposed ALER, a memory-augmented RL agent, together with rune mazes, in which runes change the rule that maps an early cue to the target. To analyze these tasks, we formalized retention, rewriting, and experience fusion as three classes of target updates and derived from the rune automaton of a task how much memory a composition of rules requires. ALER assigns the three classes to separate components. An LSTM carries retention, an independently addressed slot memory carries rewriting, and a learned gate fuses the retrieved content with the recurrent state. With this division, ALER reaches a success rate of at least 0.82 in every Endless T-Maze configuration, keeps a success rate of 1.00 on corridors up to three times the training length, reaches at least 0.99 on every Rune T-Maze composition, and leads PPO-LSTM in eight of ten Rune MiniGrid Memory configurations. Probes show that the learned memory follows the deferred solution of our composition result. The slot memory stores a new cue with one write and decodes the initial cue, and the recurrent state decodes the rune flags. Rune Multi-Corridor with $K \geq 8$ remains open for all eight architectures.

Reproducibility Statement

Section 4 and Appendix B specify ALER, including addressing noise, temperatures, and initialization. Appendix C specifies the observations, actions, rewards, and rune semantics of every environment. Section 6 gives the training seeds, the checkpoint selection rule, and the evaluation seeds, and Appendix F lists all hyperparameters.

AI USE STATEMENT

Generative AI tools were used solely to aid and polish the writing of the paper. Details are provided in the paper.

REFERENCES

Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. Deepmind lab, 2016. URL https://arxiv.org/abs/1612.03801.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. doi: 10.1613/jair.3912.

Semih Cayci and Atilla Eryilmaz. Recurrent natural policy gradient for pomdps, 2024. URL https://arxiv.org/abs/2405.18221. Transactions on Machine Learning Research.

Hugo Chateau-Laurent and Frédéric Alexandre. Relating hopfield networks to episodic control. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 99921–99946. Curran Associates, Inc., 2024. doi: 10.52202/079017-3170. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/b528459c99e929718a7d7e1697253d7f-Paper-Conference.pdf.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Egor Cherepanov, Nikita Kachaev, Alexey Kovalev, and Aleksandr Panov. Memory, benchmark &; robots: A benchmark for solving complex tasks with reinforcement learning. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *Memory, Benchmark & Robots: A Benchmark for Solving Complex Tasks with Reinforcement Learning*, volume 2026, pp. 9759–9815, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/1088fffae5f2aa8c8324e4f45f62248d-Paper-Conference.pdf.

Egor Cherepanov, Alexey Kovalev, and Aleksandr Panov. Elmur: External layer memory with update/rewrite for long-horizon rl problems. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 141814–141844, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/e58dfd0157cb9de91e7a513d3148aad5-Paper-Conference.pdf.

Egor Cherepanov, Aleksei Staroverov, Alexey Kovalev, and Aleksandr Panov. Recurrent action transformer with memory. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 129379–129407, 2026c. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/d21b249204cace583de6beabcb6e4b3d-Paper-Conference.pdf.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks, 2023. URL https://arxiv.org/abs/2306.13831.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning, 2020. URL https://arxiv.org/abs/1912.01588.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context, 2019. URL https://arxiv.org/abs/1901.02860.

Kevin Esslinger, Robert Platt, and Christopher Amato. Deep transformer q-networks for partially observable reinforcement learning, 2022. URL https://arxiv.org/abs/2206.01078.

Noah Farr, Aryaman Reddi, Carlo D'Eramo, and Jan Peters. Streaming reinforcement learning under partial observability with real-time recurrent learning, 2026. URL https://arxiv.org/abs/2605.24709.

Meire Fortunato, Melissa Tan, Ryan Faulkner, Steven Hansen, Adrià Puigdomènech Badia, Gavin Buttimore, Charlie Deck, Joel Z Leibo, and Charles Blundell. Generalization of reinforcement learners with working and episodic memory, 2019. URL https://arxiv.org/abs/1910.13406.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014. URL https://arxiv.org/abs/1410.5401.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

Caglar Gulcehre, Sarath Chandar, and Yoshua Bengio. Memory augmented neural networks with wormhole connections, 2017. URL https://arxiv.org/abs/1701.08718.

Gunshi Gupta, Karmesh Yadav, Zsolt Kira, Yarin Gal, and Rahaf Aljundi. Memo: Training memory-efficient embodied agents with reinforcement learning. *arXiv preprint arXiv:2510.19732*, 2025.

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

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax, 2017. URL https://arxiv.org/abs/1611.01144.

Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, Jonathan Harper, Ervin Teng, Hunter Henry, Adam Crespi, Julian Togelius, and Danny Lange. Obstacle tower: A generalization challenge in vision, control, and planning, 2019. URL https://arxiv.org/abs/1902.01378.

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.

Andrew K. Lampinen, Stephanie C. Y. Chan, Andrea Banino, and Felix Hill. Towards mental time travel: A hierarchical memory for reinforcement learning agents, 2021. URL https://arxiv.org/abs/2105.14039.

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 (ICLR)*, 2025. arXiv:2410.10132.

Chengzhi Liu, Yuzhe Yang, Sophia Xiao Pu, Yepeng Liu, Lin Long, Yichen Guo, Nuo Chen, Zhaotian Weng, Elena Kochkina, Simerjot Kaur, Charese Smiley, Xiaomo Liu, James Zou, Sheng Liu, Yuheng Bu, Songyou Peng, and Xin Eric Wang. Worldmemarena: Evaluating multimodal agent memory through action-world interaction. *arXiv preprint arXiv:2605.29341*, 2026.

Chris Lu, Yannick Schroecker, Albert Gu, Emilio Parisotto, Jakob Foerster, Satinder Singh, and Feryal Behbahani. Structured state space models for in-context reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 47016–47031, 2023. doi: 10.52202/075280-2036.

Lina Mezghani, Sainbayar Sukhbaatar, Thibaut Lavril, Oleksandr Maksymets, Dhruv Batra, Piotr Bojanowski, and Karteek Alahari. Memory-augmented reinforcement learning for image-goal navigation. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3316–3323, 2022.

Steven Morad, Ryan Kortvelesy, Matteo Bettini, Stephan Liwicki, and Amanda Prorok. Popgym: Benchmarking partially observable reinforcement learning, 2023a. URL https://arxiv.org/abs/2303.01859.

Steven Morad, Ryan Kortvelesy, Stephan Liwicki, and Amanda Prorok. Reinforcement learning with fast and forgetful memory. *Advances in Neural Information Processing Systems*, 36: 72008–72029, 2023b.

Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent model-free rl can be a strong baseline for many pomdps, 2022. URL https://arxiv.org/abs/2110.05038.

Tianwei Ni, Michel Ma, Benjamin Eysenbach, and Pierre-Luc Bacon. When do transformers shine in rl? decoupling memory from credit assignment, 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. *arXiv preprint arXiv:2210.13383*, 2022.

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.

Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control, 2017. URL https://arxiv.org/abs/1703.01988.

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

Oleg Shchendrigin, Egor Cherepanov, Alexey K. Kovalev, and Aleksandr I. Panov. Memory retention is not enough to master memory tasks in reinforcement learning. In *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 2026. doi: 10.65109/SOSN3643.

Jimmy T. H. Smith, Andrew Warrington, and Scott W. Linderman. Simplified state space layers for sequence modeling. In *International Conference on Learning Representations*, 2023.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite, 2018. URL https://arxiv.org/abs/1801.00690.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Zekang Wang, Zhe He, Borong Zhang, Edan Toledo, and Steven Morad. Investigating memory in model-free rl with popgym arcade, 2025. URL https://arxiv.org/abs/2503.01450. International Conference on Machine Learning (ICML) 2026.

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, 2018. URL https://arxiv.org/abs/1803.10760.

Yifan Wu, Lizhu Zhang, Yuhang Zhou, Mingyi Wang, Bo Peng, Serena Li, Xiangjun Fan, and Zhuokai Zhao. Remember when it matters: Proactive memory agent for long-horizon agents. *arXiv preprint arXiv:2607.08716*, 2026.

Zhao Yang, Thomas M. Moerland, Mike Preuss, and Aske Plaat. Two-memory reinforcement learning. In *2023 IEEE Conference on Games (CoG)*, pp. 1–9. IEEE, 2023. doi: 10.1109/CoG57401.2023.10333174.

Wojciech Zaremba and Ilya Sutskever. Reinforcement learning neural turing machines - revised, 2015. URL https://arxiv.org/abs/1505.00521.

Zeyu Zhang, Quanyu Dai, Rui Li, Xiaohe Bo, Xu Chen, and Zhenhua Dong. Learn to memorize: Optimizing llm-based agents with adaptive memory framework. *arXiv preprint arXiv:2508.16629*, 2025.

Appendix Contents

- Proofs 13
- ALER Detailed Description 15
- Rune-Mazes Detailed Description 16
- Additional Results 19
- Mechanism Analysis Details 21
- Hyperparameters 24
- Compute and LLM Usage 27

A Proofs

A.1 Proof of Proposition 1

Let the action at τ follow a possibly randomized rule $\pi(a \mid m_\tau, o_\tau)$. Then

$$P(a_\tau = y_\tau) = \sum_{m,o,y} P(m,o,y)\,\pi(y \mid m,o) \le \sum_{m,o} \max_y P(m,o,y) \le \sum_m \sum_o \max_y P(o,y).$$

By (D), $P(o,y) = P(o)/K$ for every y, so the inner sum equals $1/K$ and the total is at most M/K. A probability is also at most one. For continuous observations, the sum over o becomes an integral with the same bound. The step τ may depend on the policy. In our environments, the agent crosses every rune before the junction, o_τ is a function of the rune placement alone, and y_τ is the cue transformed by the runes of the episode. For each placement, y_τ is a bijective function of c. The cue is uniform and independent of the placement, so y_τ is uniform and independent of o_τ for every policy. □

A.2 Proof of Proposition 2

Let $T(c,\sigma)$ be the target after cue c and rune sequence $\sigma \in \Sigma^*$. Call two prefixes (c,σ) and (c',σ') equivalent if $T(c,\sigma u) = T(c',\sigma' u)$ for every continuation $u \in \Sigma^*$. The states of the minimal automaton for T are exactly these equivalence classes, so there are $Q^\star(\Sigma)$ of them.

(a) We use the corridor of the statement and index the episode by corridor position. The update f and the action map are the same at every step. Take two prefixes (c, σ) and (c', σ'), place the runes of both before a common position p, and place a continuation u at the same positions after p in both episodes. After the first step, the cue is masked, and each observation is determined by the position and the rune crossed at that step. Suppose that the memory values are equal at the first step at which the agent is at p in the two episodes. From that step on, both episodes receive the same observations and take the same actions, including turns that do not move the agent, so they reach the junction the same number of steps after p with the same memory and the same decision. The hypothesis requires a decision in every episode, so truncation does not occur. If the prefixes are inequivalent, some continuation u gives them different targets, and the agent errs on one of the two episodes. Hence prefixes from different classes reach different memory values at p. Every class contains a prefix, and the corridor lets all these prefixes end before the same p, so at least $Q^\star(\Sigma)$ memory values are reachable at p. Consider next an agent whose update or action uses noise that is independent and identically distributed across steps, such as Gumbel noise, and that is correct with probability one. The first step at p is a stopping time, so the noise after it is independent of the memory at p and has the same law in both episodes. For inequivalent prefixes A and B with separating continuation u, let S_A be the set of memory values from which the agent, given u, selects the target of A with probability one, and define S_B likewise. The memory at p lies in S_A with probability one in the episode of A and in S_B with probability one in the episode of B, and $S_A \cap S_B = \emptyset$ because the two targets differ. The laws of the memory at p are therefore concentrated on disjoint sets for inequivalent prefixes. Take one prefix from each class. Each prefix A reaches a value that lies in S_A for every separating continuation, these values differ across the chosen prefixes, and at least $Q^\star(\Sigma)$ values are therefore reachable at p.

(b) The deferred solution keeps the cue c and the state q of the minimal rune automaton. Its output is the target, $c \oplus \mathrm{inv}(q)$ in Rune T-Maze and $\mathrm{inv}(q)$ applications of $y \mapsto K - 1 - y$ to c in Rune Multi-Corridor. It uses $KQ(\Sigma)$ states and computes T, and the minimal automaton has at most as many states. □

A.3 Computing Q and $Q^\star$

The state of the rune automaton of Rune T-Maze is $(\mathit{invert}, \mathit{skip}, \mathit{last})$ with $\mathit{invert}, \mathit{skip} \in \{0, 1\}$ and *last* the last rune that took effect. The transitions follow Section C.1. `Invert` flips *invert*, `SkipNext` sets *skip*, `ResetRules` clears both flags, and `NoOp` changes neither. A rune that arrives while $\mathit{skip} = 1$ clears *skip* and leaves *last* unchanged. `RepeatPrev` re-applies *last* if it is a rune other than `RepeatPrev` and then sets *last* to `RepeatPrev`. For each composition, we enumerate the states reachable from $(0, 0, \text{none})$ over its alphabet and merge equivalent states by partition refinement. With output *invert* this gives Q. With the cue added to the state and output $c \oplus \mathit{invert}$ it gives $Q^\star$. For `3 Invert + ResetRules`, the cue and the invert flag are both needed after a reset, which gives $Q^\star = 4 = 2Q$. For the other four compositions, the minimal target automaton merges the cue into the invert flag, which gives $Q^\star = Q$. A sampled episode contains each rune candidate of its composition at most once, in uniformly random order, so the rune sequences that occur form a finite set. The candidates are one `Invert` for `Invert`, `NoOp` and `Invert` for `NoOp + Invert`, `SkipNext` and `Invert` for `SkipNext + Invert`, three `Invert` and one `ResetRules` for `3 Invert + ResetRules`, and `RepeatPrev`, `SkipNext`, and `Invert` for `RepeatPrev + SkipNext + Invert`. Each candidate is present with probability 0.8, and every order of the present candidates fits the corridor of length 10, so every such sequence occurs with positive probability. On this set we search for the smallest deterministic update that selects the correct branch on every sampled episode, by coloring the tree of cue and rune prefixes so that equal colors have equal outputs and equal successors. $Q^\star_{\mathrm{ep}}$ is the number of colors after the cue. Q_{ep} is the number of colors in the same search over rune prefixes alone with output *invert*, and it includes the initial state, as Q does. This search gives $Q^\star_{\mathrm{ep}} = 2, 2, 4, 4, 6$ and $Q_{\mathrm{ep}} = 2, 2, 3, 2, 4$ for `Invert`, `NoOp + Invert`, `SkipNext + Invert`, `3 Invert + ResetRules`, and `RepeatPrev + SkipNext + Invert`. The values of $Q^\star_{\mathrm{ep}}$ also bound the memory of an agent from below. Let n be the number of candidates. For $s \le n$, let G_s be the graph whose nodes are the cue and rune prefixes with at most s runes and whose edges join two prefixes if a continuation of length at most $n - s$ exists that uses only candidates left unused by each of the two prefixes and gives them different targets. Rune positions have positive density in $(0.02, 0.98)$ at a spacing of at least $0.1 = 1/L$, so each step crosses at most one rune, and observations depend

Algorithm 4 ALER forward step.

Require: Observation o_t, memory M_{t-1}, recurrent state (h_{t-1}, c_{t-1}), episode-start flag e_t
if $e_t = 1$ **then**
  Set (h_{t-1}, c_{t-1}) and M_{t-1} to zero
end if
$(h_t, c_t) \leftarrow \mathrm{LSTM}(\phi(o_t), h_{t-1}, c_{t-1})$
$r_t \leftarrow \texttt{Read}(h_t, M_{t-1}, \tau_r)$
$z_t \leftarrow \texttt{Fusion}(h_t, r_t)$
$M_t \leftarrow \texttt{Write}(h_t, M_{t-1}, \tau_w)$
$\pi_t \leftarrow \mathrm{Actor}(\mathrm{MLP}_\pi(z_t)), \quad V_t \leftarrow \mathrm{Critic}(\mathrm{MLP}_V(z_t))$
Sample $a_t \sim \pi_t$ in training, take $a_t = \arg\max_a \pi_t(a)$ in evaluation
return a_t, V_t, $\log \pi_t(a_t)$, M_t, (h_t, c_t)

on a placement only through the step at which each rune is crossed. Placing both prefixes before a common position p and the continuation after p therefore gives, for each of the two episodes, a set of placements of positive probability with the same crossing steps. By the argument of Proposition 2(a), an agent that selects the correct branch with probability one needs different memory values at p for adjacent prefixes, and hence at least $\chi(G_s)$ memory values. For all five compositions, the computed $\max_s \chi(G_s)$ equals $Q^\star_{\mathrm{ep}}$, so this lower bound is tight. The minimal automata over Σ^* restricted to the sampled prefixes are valid colorings, so the counts over Σ^* bound the sampled counts from above, and both counts order the compositions by $Q^\star$ in the same way.

A.4 Proof of Proposition 3

(a) By induction on t. The case $t = 0$ holds with $M_0 = 0$. If (a) holds at $t-1$, substituting it into $m_{t,i} = (1 - w^w_{t,i}) m_{t-1,i} + w^w_{t,i} x_t$ multiplies every earlier coefficient by $(1 - w^w_{t,i})$ and adds x_t with coefficient $w^w_{t,i}$. (b) The coefficients in (a) are nonnegative and sum to $1 - \prod_{u=1}^{t}(1 - w^w_{u,i}) \le 1$, so $m_{t,i}$ is a convex combination of $0, x_1, \ldots, x_t$, and the triangle inequality gives the bound. (c) $m_{t,i} - x_t = (1 - w^w_{t,i})(m_{t-1,i} - x_t)$, and $1 - w^w_{t,i} \le \varepsilon$. Since w^w_t lies on the simplex, $\sum_{j \ne i} w^w_{t,j} = 1 - w^w_{t,i} \le \varepsilon$. □

B ALER Detailed Description

This section gives the pseudocode of one ALER step in the notation of Section 4. The memory and the LSTM state are zero at the start of every episode, and $\mathrm{normalize}(v) = v / \max(\|v\|, 10^{-8})$, so $\mathrm{normalize}(0) = 0$. At the first step, all read and write logits are therefore zero, the retrieved vector is $r_1 = 0$, and the first write is spread over slots by the Gumbel noise alone. Algorithms 1–3 specify the three memory operations, and Algorithm 4 places them in the actor-critic step.

Algorithm 1: Read

Require: h_t, M_{t-1}, τ_r
$q^r_t \leftarrow \mathrm{normalize}(W_Q h_t)$
for $i = 1$ to C **do**
  $k^r_{t,i} \leftarrow \mathrm{normalize}(W_K m_{t-1,i})$
  $s_{t,i} \leftarrow (q^r_t)^\top k^r_{t,i}$
end for
$w^r_t \leftarrow \mathrm{GumbelSoftmax}(s_t, \tau_r)$
$r_t \leftarrow \sum_i w^r_{t,i} W_V m_{t-1,i}$
return r_t

Algorithm 2: Fusion

Require: h_t, r_t
$g_t \leftarrow \sigma(W_g[h_t, r_t] + b_g)$
$z_t \leftarrow g_t \odot r_t + (1 - g_t) \odot h_t$
return z_t

Algorithm 3: Write

Require: h_t, M_{t-1}, τ_w
$x_t \leftarrow W_{\mathrm{write}} h_t + b_{\mathrm{write}}$
for $i = 1$ to C **do**
  $\ell^w_{t,i} \leftarrow \frac{(W_{Q_w} x_t)^\top W_{K_w} m_{t-1,i}}{\sqrt{d_k}}$
end for
$w^w_t \leftarrow \mathrm{GumbelSoftmax}(\ell^w_t, \tau_w)$
for $i = 1$ to C **do**
  $m_{t,i} \leftarrow (1 - w^w_{t,i}) m_{t-1,i} + w^w_{t,i} x_t$
end for
return M_t

Components. A flat feature extractor maps the observation to $\phi(o_t)$, and a single-layer LSTM with hidden size 256 processes it. The actor and critic have separate MLP heads with two hidden layers of 256 units, and both receive the fused representation z_t. The read projections are $W_Q, W_K \in \mathbb{R}^{d_k \times d}$ and $W_V \in \mathbb{R}^{d \times d}$, and the gate parameters are $W_g \in \mathbb{R}^{d \times 2d}$ and $b_g \in \mathbb{R}^d$. Read scores are cosine similarities between the normalized query and keys. All C slots take part in reading and writing from the first step, including slots whose content is still zero.

Addressing in training and evaluation. Both addressing steps use the relaxed Gumbel-Softmax weights of Section 4 with noise $\gamma_i = -\log(-\log u_i)$ and $u_i \sim \mathcal{U}(10^{-6}, 1 - 10^{-6})$ in training and in evaluation, with fresh noise at every step and separate noise for reading and writing. The noise is part of the policy. The evaluation seed sets both the episodes and the noise stream, so the validation episodes with seed 123 use one fixed noise stream, and the SEM over training seeds reflects variation across trained agents. Evaluation seeds 1–3 give new episodes and new noise, and the ALER agents of Section 8 keep 1.00 ± 0.00 on `Invert` and `RepeatPrev + SkipNext + Invert` there (Table 10). Gradients flow to every slot in proportion to its write weight. The temperatures stay fixed during training, and Table 11 lists them for each environment.

C Rune-Mazes Detailed Description

This section specifies the observations, actions, rewards, and rune semantics of the three Rune-Mazes environments. Table 6 lists the trained configurations.

Table 6: **Training and evaluation configurations for Rune T-Maze and Rune Multi-Corridor.** Each row is a separate training task.

Environment	Configuration
Rune T-Maze	`Invert`, `NoOp + Invert`, `SkipNext + Invert`, `3 Invert + ResetRules`, `RepeatPrev + SkipNext + Invert`
Rune Multi-Corridor	$K \in \{4, 8, 16\}$, no runes
Rune Multi-Corridor	$K \in \{4, 8, 16\}$, one `Invert` rune

C.1 Rune T-Maze

Rune T-Maze modifies the classic T-Maze. At the start of each episode, the agent is at position $\xi = 0$ of a one-dimensional corridor of normalized length 1 and moves toward the junction at $\xi = 1$. A binary cue, with 0 for left and 1 for right, indicates the initially correct turn. It is drawn uniformly and independently of the rune placement. The cue is visible only in the first observation and is masked to -1 afterwards.

Observations and actions. At each step, the agent receives the vector $o_t = [junc, c, dir, rune]$. Here $junc \in \{0, 1\}$ equals one at the junction, $c \in \{0, 1, -1\}$ is the cue, dir is the movement direction, and $rune$ is the identifier of the rune just crossed. The identifiers $0, 1, 3, 4, 5$ denote `Invert`, `SkipNext`, `ResetRules`, `RepeatPrev`, and `NoOp`, and $rune = -1$ marks a step without a rune. The coordinate ξ is not observed, and the direction stays $dir = 1$ in the evaluated compositions. The agent has three actions, forward, turn left, and turn right. A turn has an effect only at the junction, $\xi \geq 1$. Elsewhere, it costs the step penalty and does not end the episode.

Corridor and rune placement. The corridor length is $L = L_{\max}$ in fixed mode and $L \sim \mathcal{U}[L_{\min}, L_{\max})$ in uniform mode, and each forward step advances the position by $1/L$. The evaluated configurations use $L = 10$. Before each episode, per-type counts specify candidate runes, whose order is randomized and whose positions are drawn in $(0.02, 0.98)$ with a minimum spacing of 0.1 between candidates. Each candidate appears independently with probability $p = 0.80$. A rune is consumed when crossed and cannot fire again.

Rune semantics. The environment keeps an *invert* flag, a *skip* flag, and the last rune that took effect. `Invert` flips the invert flag. `SkipNext` sets the skip flag. When a rune arrives while the

skip flag is set, the rune is consumed without effect, the skip flag is cleared, and the last effective rune stays unchanged. `ResetRules` clears both flags and sets the direction forward. `RepeatPrev` reapplies the last effective rune, acts as `NoOp` if no rune has taken effect or the last effective rune is `RepeatPrev`, and then becomes the last effective rune. Every other rune that takes effect also becomes the last effective rune. `NoOp` changes neither flag and becomes the last effective rune. At the junction, turning right is correct if and only if $c \oplus \mathit{invert} = 1$, where c is the cue and invert the invert flag.

Rewards. A correct turn at the junction gives $+1$ and an incorrect turn -1, and both end the episode. Every other step, including forward moves and turns outside the junction, gives -0.01. Episodes are truncated after $T_{\max} = 200$ steps.

C.2 Rune Multi-Corridor

Rune Multi-Corridor extends Rune T-Maze to K branches. The agent starts at integer position 0 of a corridor of L steps, walks to the junction at position L, and selects one of K branches. The target branch index is drawn uniformly from $\{0, \dots, K-1\}$ independently of the rune placement and is encoded as a vector of $\lceil \log_2 K \rceil$ bits, which forms the cue in the first observation and is masked to -1 afterwards.

Observations and actions. At each step, the agent receives $o_t = [\mathit{junc}, \mathbf{b}, \mathit{rune}]$ of dimension $2 + \lceil \log_2 K \rceil$. Here junc equals one at the junction, $\mathbf{b} \in \{0, 1, -1\}^{\lceil \log_2 K \rceil}$ is the cue, and $\mathit{rune} \in \{-1, 0\}$ marks an `Invert` rune at the current step. The coordinate is not observed. The agent has $1 + K$ actions. Action 0 moves forward, and action $1 + k$ selects branch k. Branch selections have an effect only at the junction and cost the step penalty elsewhere.

Corridor, runes, and rewards. The corridor length is $L = L_{\max}$ in fixed mode and $L \sim \mathcal{U}\{L_{\min}, \dots, L_{\max}\}$ in uniform mode, and episodes are truncated after $T_{\max} = 2L$ steps. All evaluated configurations use fixed $L = 10$. In the `Invert` configurations, one candidate rune is placed uniformly at an interior position $\{1, \dots, L-1\}$ and appears with probability $p = 0.80$. The no-rune configurations have no candidates. Crossing the rune inverts all bits of the current target, $y \leftarrow (2^{\lceil \log_2 K \rceil} - 1) - y$, which equals $K - 1 - y$ for the evaluated K. For example, with $K = 8$ and cue 5 (binary 101), one `Invert` rune changes the target to 2 (binary 010). The agent succeeds if it selects the current target at the junction. Rewards follow Rune T-Maze.

C.3 Rune MiniGrid Memory

Rune MiniGrid Memory adds the runes of Rune T-Maze to MiniGrid Memory (Chevalier-Boisvert et al., 2023). It keeps the original action space, partial RGB observations, reward function, termination checks, and cue generation. The agent starts in the cue room facing east, and random-length maps are resampled until the requested runes fit with the required spacing. Runes lie only on corridor cells that every path to the junction crosses, so the agent always observes the cue, then the runes, then the junction.

Runes are passable floor tiles placed after the cue room and at least two cells apart. A rune can be visible before contact, fires when the agent enters its cell, is consumed, and disappears from the grid. Their semantics follow Rune T-Maze and are listed in Table 7. We evaluate the five Rune T-Maze compositions with fixed and random corridor lengths and train each model for 30M environment steps with three training seeds. Figure 5 shows one rollout per composition.

Table 7: **Rune legend for Rune MiniGrid Memory.** Each rune is a floor tile of its own color.

Tile	Rune (color)	Effect
	`Invert` (purple)	Swaps the hidden success and failure targets. The two visible objects stay in place.
	`SkipNext` (yellow)	Cancels the next rune the agent steps on. That rune is consumed without effect.
	`ResetRules` (green)	Restores the original targets, clears the `Invert` and `SkipNext` effects, and turns the agent east.
	`RepeatPrev` (red)	Re-applies the last rune that took effect and has no effect if no rune has fired.
	`NoOp` (grey)	Leaves the rules unchanged and appears as a visual distractor.

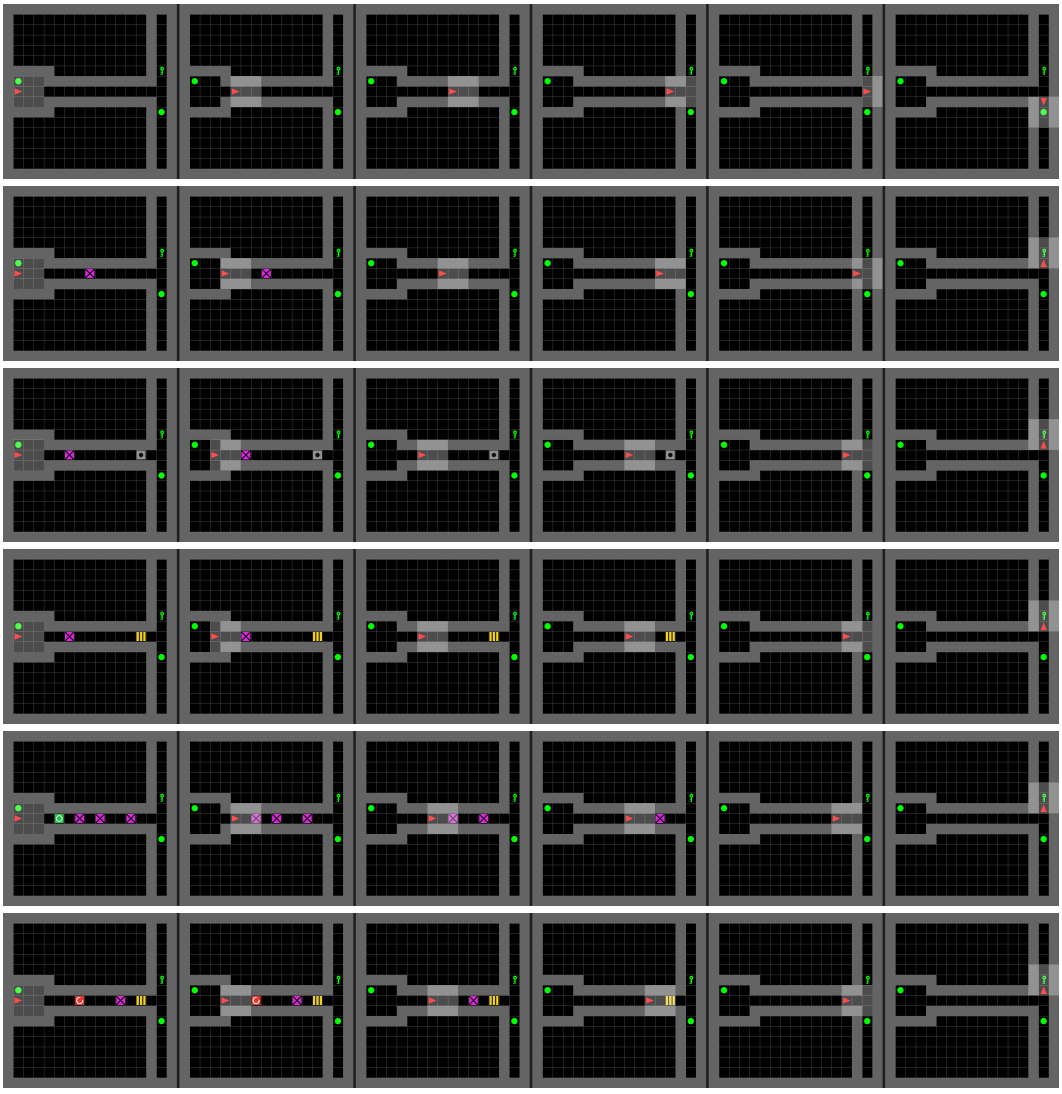

Figure 5: **Example Rune MiniGrid Memory rollouts.** From top to bottom: no runes, `Invert`, `NoOp + Invert`, `SkipNext + Invert`, `3 Invert + ResetRules`, and `RepeatPrev + SkipNext + Invert`. Each strip follows one episode from the cue to the junction.

D Additional Results

This section reports Endless T-Maze with random corridor lengths, the retention-only MiniGrid Memory results cited in Section 7, and the learning curves of the vector environments.

D.1 Endless T-Maze with random corridor lengths

Table 8 completes Table 2 with the eight configurations in which every corridor has a random length. ALER and S5 reach 1.00 in all eight, and PPO-LSTM reaches 1.00 in seven and 0.98 ± 0.02 in the remaining one.

Table 8: **Endless T-Maze with random corridor lengths.** Each of the n corridors has a length sampled uniformly from the range l. Success rate of the selected checkpoint on the 100 validation episodes, mean $\pm$ SEM over five training seeds. Bold marks every value equal to the best mean in its row.

l	n	PPO-MLP	PPO-LSTM	GTrXL	S5	FFM	SHM	DNC	ALER
2–5	1	0.26 ± 0.02	$\mathbf{1.00 \pm 0.00}$	0.47 ± 0.08	$\mathbf{1.00 \pm 0.00}$	0.92 ± 0.08	0.55 ± 0.23	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$
2–5	3	0.01 ± 0.01	$\mathbf{1.00 \pm 0.00}$	0.15 ± 0.09	$\mathbf{1.00 \pm 0.00}$	0.43 ± 0.23	0.04 ± 0.01	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$
2–5	5	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$	0.04 ± 0.01	$\mathbf{1.00 \pm 0.00}$	0.03 ± 0.03	0.04 ± 0.02	0.02 ± 0.01	$\mathbf{1.00 \pm 0.00}$
2–5	10	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$
2–10	1	0.24 ± 0.12	$\mathbf{1.00 \pm 0.00}$	0.57 ± 0.02	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.91 ± 0.09	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$
2–10	3	0.00 ± 0.00	0.98 ± 0.02	0.16 ± 0.03	$\mathbf{1.00 \pm 0.00}$	0.00 ± 0.00	0.06 ± 0.04	0.02 ± 0.01	$\mathbf{1.00 \pm 0.00}$
2–10	5	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$	0.06 ± 0.01	$\mathbf{1.00 \pm 0.00}$	0.00 ± 0.00	0.04 ± 0.01	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$
2–10	10	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$	0.01 ± 0.01	$\mathbf{1.00 \pm 0.00}$	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$

D.2 MiniGrid Memory

Before adding runes, we compare ALER and PPO-LSTM on the retention-only MiniGrid Memory tasks (Chevalier-Boisvert et al., 2023) in all eight configurations, S9, S11, S13, and S17 and their random variants. ALER has the higher mean success rate in seven of the eight configurations and an average success rate of 0.84, against 0.69 for PPO-LSTM (Table 9).

Table 9: **MiniGrid Memory.** Success rate of the selected checkpoint on the 100 validation episodes, mean $\pm$ SEM over three training seeds. Bold marks the higher mean in each row.

Configuration	PPO-LSTM	ALER
S9	0.66 ± 0.13	$\mathbf{0.84 \pm 0.22}$
S9Random	0.83 ± 0.22	$\mathbf{1.00 \pm 0.00}$
S11	0.55 ± 0.01	$\mathbf{0.67 \pm 0.19}$
S11Random	0.84 ± 0.23	$\mathbf{0.95 \pm 0.07}$
S13	0.55 ± 0.00	$\mathbf{0.83 \pm 0.23}$
S13Random	0.78 ± 0.16	$\mathbf{0.97 \pm 0.04}$
S17	$\mathbf{0.55 \pm 0.00}$	0.50 ± 0.05
S17Random	0.72 ± 0.15	$\mathbf{0.98 \pm 0.03}$

D.3 Learning curves

Figures 6 and 7 show the validation return and success rate over training on Rune Multi-Corridor and Rune T-Maze. Curves show the mean over training seeds, and shaded regions show the SEM.

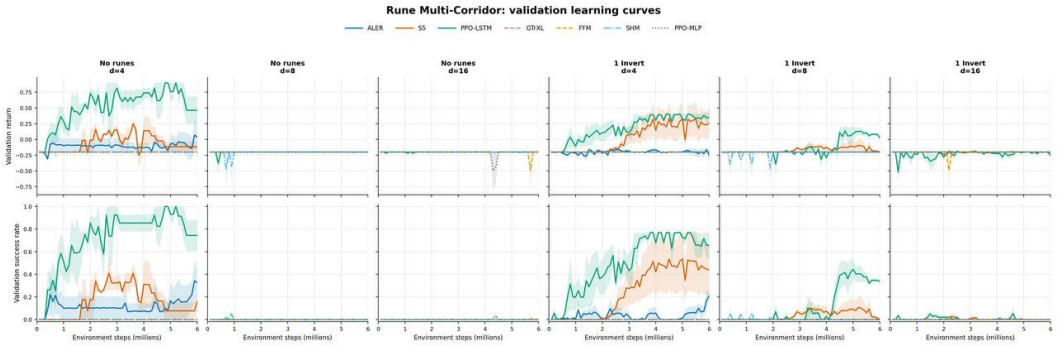

Figure 6: **Rune Multi-Corridor learning curves.** Validation return and success rate over training for ALER, S5, PPO-LSTM, GTrXL, FFM, SHM, and PPO-MLP for $K \in \{4, 8, 16\}$ branches, with (`rune=1`) and without (`rune=0`) an `Invert` rune.

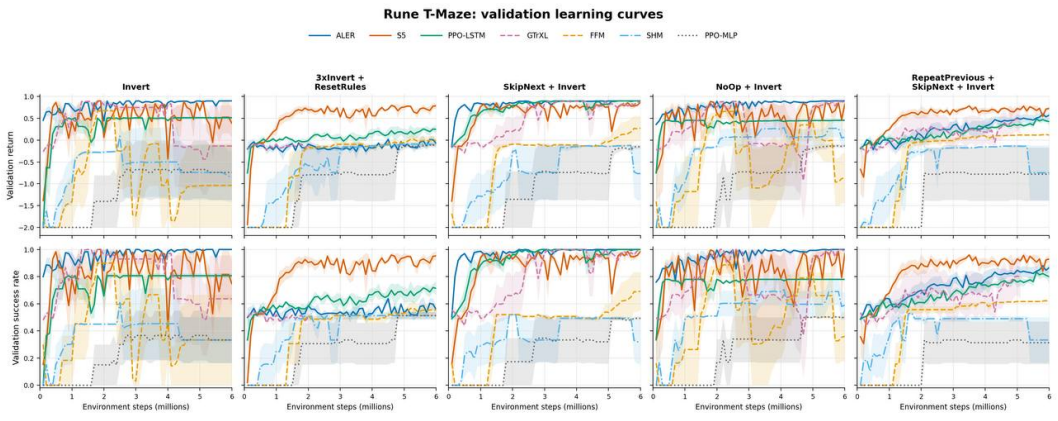

Figure 7: **Rune T-Maze learning curves.** Validation return and success rate over training for ALER, S5, PPO-LSTM, GTrXL, FFM, SHM, and PPO-MLP. Column titles give the per-type rune counts of each composition.

E MECHANISM ANALYSIS DETAILS

This section gives the protocols and full results behind Section 8.

E.1 EVALUATION-TIME GATE EXPERIMENTS

The gate $g_t \in [0,1]^{256}$ has one coefficient per feature. We replace it at every evaluation step by the constant 0, 0.5, or 1, with frozen weights and active memory writes. These settings give the policy the recurrent state alone, an equal mixture, or the retrieved vector alone. On Endless T-Maze, we use the checkpoint trained without the coordinate input that also serves the probes (Section E.2). Equal mixing lowers the success rate to 0.06, 0.45, and 0.54 on the three tasks of Table 10, and the retrieved vector alone gives 0.00, 0.26, and 0.21. A constant mixing ratio thus lowers success on all three tasks, and the learned feature-wise gate gives the highest success.

Table 10: **Gate experiments on trained ALER.** Success rate, mean $\pm$ SEM over three training seeds, on evaluation seeds 1–3 with 100 episodes each. ET is Endless T-Maze with 20 corridors of length 50, and RT-1 and RT-2 are Rune T-Maze `Invert` and `RepeatPrev + SkipNext + Invert`.

Gate	ET	RT-1	RT-2
Learned g_t	$\mathbf{0.72 \pm 0.14}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$
$g_t = 0$	0.48 ± 0.29	0.83 ± 0.02	0.72 ± 0.04
$g_t = 0.5$	0.06 ± 0.06	0.45 ± 0.11	0.54 ± 0.13
$g_t = 1$	0.00 ± 0.00	0.26 ± 0.13	0.21 ± 0.11

E.2 LINEAR PROBES

We fit standardized ridge classifiers ($\alpha = 10$) separately for each training seed, representation, and decoded variable, on 128 episodes per checkpoint and configuration. The agent acts greedily in these episodes, and the Gumbel noise of memory addressing stays active. Whole episodes are split $75/25$, and standardization uses only the training split.

Representations. We probe the recurrent state h_t before fusion, the write candidate x_t, the memory before and after the write, M_t^- and M_t^+, the retrieved vector r_t, the gate g_t, and the fused representation z_t. We also probe the update of the dominant write slot, $\Delta m_t = M_t^+[j_t] - M_t^-[j_t]$ with $j_t = \arg\max_j w_{t,j}^w$, which measures the change of one slot. M_t^+ contains all retained and newly written content.

Endless T-Maze. For probing, we train ALER without the coordinate input on Endless T-Maze with 10 corridors of lengths sampled uniformly from 2 to 10 and select the checkpoint on 20 corridors of length 20. Figure 8 reports current- and previous-cue decoding at junctions for corridor lengths 20 to 50. Each training seed contributes one frozen checkpoint, and only the probe is refitted per length. At length 50, the current cue is decoded with 1.00 ± 0.00 from h_t, x_t, Δm_t, and z_t, with 0.98 ± 0.01 from M_t^-, 0.99 ± 0.01 from M_t^+, and 0.60 ± 0.01 from r_t. The previous cue is decoded with 0.98 ± 0.01 from h_t, 0.70 ± 0.11 from M_t^+, and 0.54 ± 0.01 from r_t. The dominant read and write slots coincide on 0.08 ± 0.01 of steps, and the normalized mutual information between dominant-slot identity and either position or current cue is below 0.01 for both operations.

Previous cue with the current cue held fixed. Consecutive cues could make the previous cue decodable through the current one. We therefore repeat junction probing on the first junction observation of corridors 2 to 20 and balance the four pairs of previous and current cue separately within the training and test splits, without filtering by success. We fit previous-cue probes separately for each value of the current cue and average their balanced accuracies. At length 50, these conditional probes reach 0.99 ± 0.01 from h_t, 0.97 ± 0.01 from x_t, and 0.98 ± 0.01 from z_t. Shuffling the previous-cue labels within each current-cue class gives 0.48 ± 0.02, 0.51 ± 0.01, and 0.51 ± 0.01, averaged over 32 shuffles per seed.

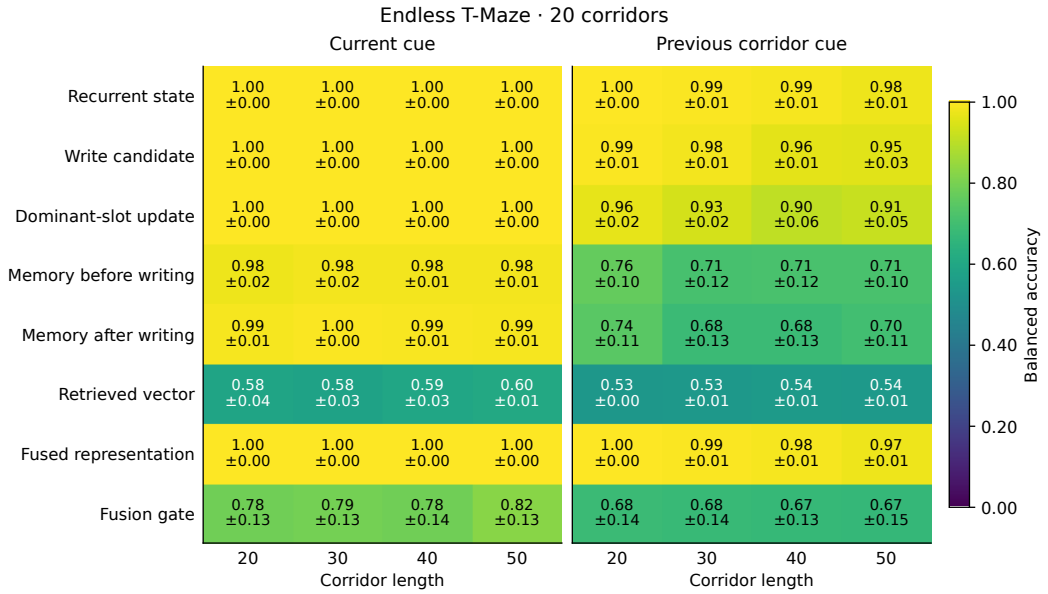

Figure 8: **Cue decoding at Endless T-Maze junctions.** Current- and previous-corridor cue decoding from eight representations with 20 corridors of the indicated length. Each probe is fitted and tested at that length on disjoint sets of episodes. Cells report balanced accuracy, mean ± SEM over three training seeds.

Cue insertion and storage within a corridor. We probe positions 0, $\mathrm{round}(L/4)$, $\mathrm{round}(L/2)$, $\mathrm{round}(3L/4)$, and L at lengths 20 to 50 with the same checkpoints and the same balancing. Only the first observation at each position is kept, so position 0 is the first presentation of a new cue. Figure 9 compares probes fitted at each position with probes fitted only at position 0. At length 50, the first cue observation raises decoding from 0.52 ± 0.01 in M_t^- to 0.99 ± 0.01 in M_t^+, and h_t, x_t, Δm_t, and z_t decode the new cue with 1.00 ± 0.00. With position-specific probes, h_t, x_t, and z_t stay at 1.00 ± 0.00 at all five positions, and M_t^+ ranges from 0.94 to 0.99. Probes fitted at position 0 reach 0.50 ± 0.00 on x_t and 0.51 ± 0.01 on M_t^+ at the junction, so the linear code of the cue changes along the corridor, and a probe fitted at the junction reads it with 1.00 ± 0.00 and 0.99 ± 0.01.

Transfer across corridor lengths. We freeze the scaler and classifier fitted at length 20 and apply them at lengths 30, 40, and 50 (Figure 10). At length 50, the current cue is decoded with 0.95 ± 0.03 from h_t, 0.95 ± 0.02 from x_t, 0.97 ± 0.02 from z_t, 0.84 ± 0.14 from M_t^+, and 0.58 ± 0.02 from r_t. The transferred previous-cue probe on h_t reaches 0.52 ± 0.01, and a probe fitted at length 50 reaches 0.98 ± 0.01, so the current-cue code stays stable across lengths and the previous-cue code changes with the interval.

Cue and rule state in Rune T-Maze. Figure 11 reports initial-cue, rule-flag, and required-turn decoding at junctions. On `Invert`, h_t, x_t, and z_t decode all three variables with 1.00 ± 0.00, and r_t decodes the required turn with 0.84 ± 0.03. On `RepeatPrev + SkipNext + Invert`, x_t decodes the initial cue with 0.99 ± 0.01 and the required turn with 0.63 ± 0.02, and the update of the dominant write slot, Δm_t, gives 0.86 ± 0.05 and 0.62 ± 0.03. From h_t, the initial cue is decoded with 0.98 ± 0.01 and the invert and skip flags with 0.70 ± 0.02 and 0.64 ± 0.03. From M_t^+, the initial cue is decoded with 0.85 ± 0.06 and the two flags with 0.59 ± 0.01 and 0.47 ± 0.09. The required turn is decoded with 0.57 ± 0.10 from M_t^+ and 0.42 ± 0.03 from r_t. Probes fitted and tested only on successful episodes raise required-turn decoding from 0.70 ± 0.04 to 0.87 ± 0.04 for h_t, from 0.63 ± 0.02 to 0.87 ± 0.00 for x_t, and from 0.64 ± 0.01 to 0.89 ± 0.04 for z_t, and r_t gives 0.51 ± 0.03. On successful episodes, the policy input z_t thus carries the required turn, and the write pathway carries the initial cue on all episodes.



Figure 9: **Current-cue decoding within Endless T-Maze corridors.** Corridors have length 50, with 20 corridors per episode. Only the first observation at each sampled position is kept, and the cue is visible at position 0 and masked at positions 12, 25, 38, and 50. Left: probes fitted at each position. Right: probes fitted at position 0 and applied without refitting. M_t^- precedes the write and M_t^+ includes it. Cells report balanced accuracy, mean ± SEM over three training seeds.

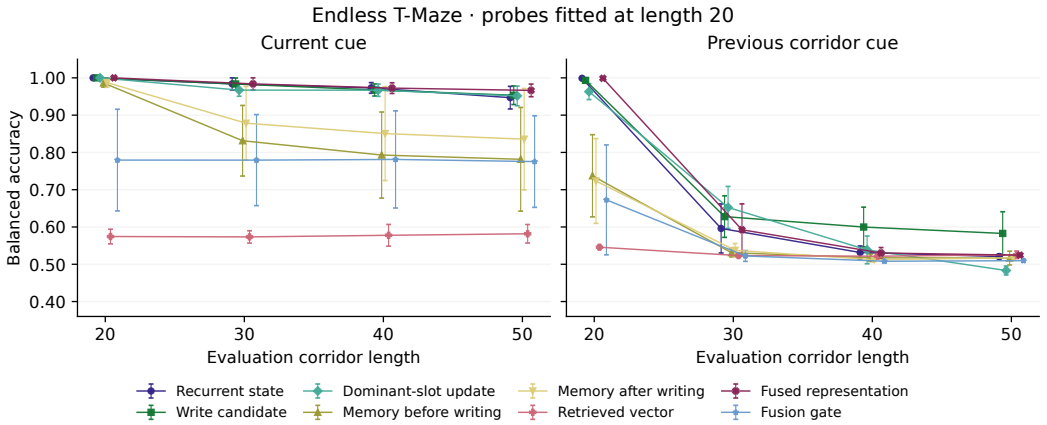

Figure 10: **Transfer of Endless T-Maze probes fitted at length 20.** Length 20 uses held-out episodes, and longer lengths use independent sets of 128 episodes per checkpoint. Points and error bars show the mean balanced accuracy and SEM over three training seeds, with small horizontal offsets between overlapping estimates.

E.3 Gate changes at rune observations

We measure the change of the mean gate coefficient and the mean absolute component change $D_t = 256^{-1} \sum_{k=1}^{256} |g_{t,k} - g_{t-1,k}|$. To control for the traversal phase, we match each rune step to non-rune steps with the same position, movement direction, and initial cue, excluding cue and junction steps. Position is used only for this matching and is not an input to the policy. We average within episodes, then across episodes for each seed. The excess of D_t over the matched controls is 0.03 ± 0.01 at `Invert` steps of the `Invert` composition and 0.02 ± 0.02, 0.04 ± 0.02, and 0.04 ± 0.01 at `Invert`, `SkipNext`, and `RepeatPrev` steps of `RepeatPrev + SkipNext + Invert`, and it is positive in each training seed for all four comparisons. Figure 12 shows the gate dynamics around rune observations.

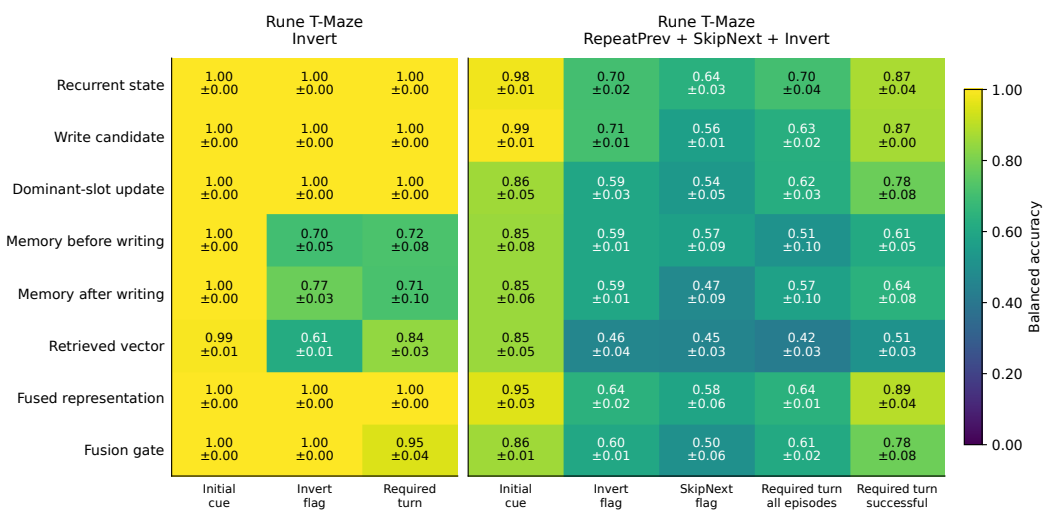

Figure 11: **Probes on Rune T-Maze.** Corridor length 10 and rune probability 0.80. All variables are decoded at junctions. The last column uses probes fitted and tested only on successful episodes, and the other columns use all episodes. Cells report balanced accuracy, mean ± SEM over three training seeds.

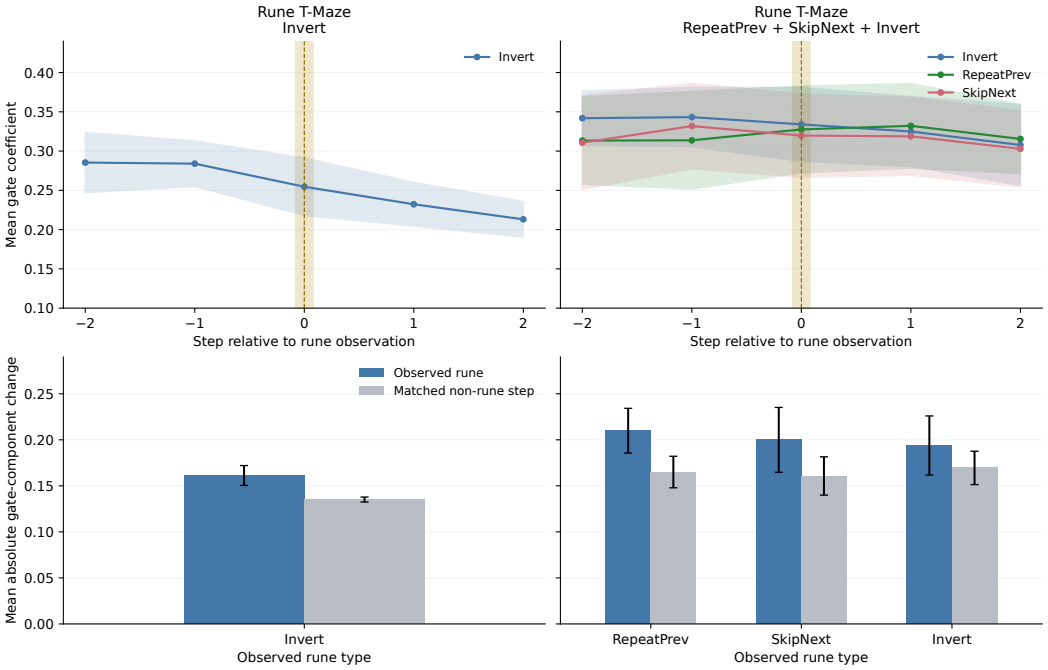

Figure 12: **Gate dynamics in Rune T-Maze.** Top: mean gate coefficient aligned to rune observations, with the rune step highlighted. Bottom: mean absolute gate-component change at rune steps and at matched non-rune steps. Shaded bands and error bars show the SEM over three training seeds.

F HYPERPARAMETERS

This section lists the selected hyperparameters of all architectures. The SHM, FFM, PPO-MLP, GTrXL, and PPO-LSTM tables describe the Rune-Mazes experiments, and their Endless T-Maze configurations follow Shchendrigin et al. (2026). Paired values list Rune T-Maze and Rune Multi-Corridor in that order. In Rune MiniGrid Memory, the policy receives MiniGrid's egocentric 56 ×

56×3 RGB view with tile size 8, and a three-layer CNN maps it to 128 features before the recurrent core. ALER uses an LSTM with 128 units, $C = 5$ slots, and $\tau_r = \tau_w = 0.2$. Both agents are trained by PPO with 32 environments with rollouts of 128 steps, minibatches of 512, 12 epochs, $\gamma = 0.99$, $\lambda = 0.98$, entropy coefficient 0.01, a learning rate annealed from $3 \cdot 10^{-4}$ to 10^{-5}, clip range 0.2, value coefficient 0.5, and gradient-norm clipping at 0.5. The grid has size 17, and validation runs every 100k steps on 100 episodes. PPO-LSTM uses the same CNN and PPO settings with an LSTM of 128 units.

Table 11: **Hyperparameters for ALER on vector-based mazes.** Tuples list Endless T-Maze, Rune T-Maze, and Rune Multi-Corridor, in that order.

Hyperparameter	Value
LSTM hidden size (d)	256
Number of LSTM layers	1
Memory capacity (C)	10
Memory slot size	256
Key size (d_k)	256
Value size	256
Read temperature (τ_r)	0.10
Write temperature (τ_w)	$(0.20, 0.10, 0.20)$
Rollout steps (n_{steps})	128
Batch size	512
PPO epochs	12
Learning rate	$(5, 3, 3) \times 10^{-4} \rightarrow 10^{-5}$
Discount (γ)	$(0.99, 0.99, 0.995)$
GAE lambda (λ)	0.98
Entropy coefficient	$(0.10, 0.01, 0.10)$
Total timesteps	6M
Framework	JAX
Parameters (vector tasks)	≈1.1M

Table 12: **Hyperparameters for SHM.**

Hyperparameter	Value
Hidden size (h)	$(512, 256)$
Memory size (m)	64
Post-processing size	1024
Learning rate	1×10^{-4}
Discount (γ)	0.99
GAE lambda (λ)	1.0
Entropy coefficient	0.001
Value loss coefficient	0.5
BPTT length	1024
Train batch size	100000
Minibatch size	8192
PPO epochs	6
Number of workers	8
Total timesteps	6M
Framework	Ray RLlib

Table 13: **Hyperparameters for FFM.**

Hyperparameter	Value
Hidden size	128
Memory size (m)	128
Post-processing size	512
Learning rate	5×10^{-5}
Discount (γ)	0.99
GAE lambda (λ)	0.95
Entropy coefficient	0.001
Value loss coefficient	0.5
BPTT length	1024
Train batch size	100000
Minibatch size	8192
PPO epochs	6
Number of workers	8
Total timesteps	6M
Framework	Ray RLlib

Table 14: **Hyperparameters for PPO-MLP.**

Hyperparameter	Value
Hidden size	128
Memory	None
Learning rate	1×10^{-4}
Discount (γ)	0.99
GAE lambda (λ)	1.0
Entropy coefficient	0.001
Value loss coefficient	0.5
BPTT length	1024
Train batch size	100000
Minibatch size	8192
PPO epochs	6
Number of workers	8
Total timesteps	6M
Framework	Ray RLlib

Table 15: **Hyperparameters for GTrXL.**

Hyperparameter	Value
Hidden size	512
Memory length	100
Transformer blocks	$(4, 2)$
Attention heads	8
Embedding dimension	512
Positional encoding	Relative
GTrXL gating	Enabled
Learning rate (initial)	$(1, 2.75) \times 10^{-4}$
Learning rate (final)	1×10^{-5}
Discount (γ)	$(0.995, 0.99)$
GAE lambda (λ)	0.95
Gradient clipping	0.25
Entropy coeff (initial)	0.001
Entropy coeff (final)	1×10^{-6}
Value loss coefficient	0.5
PPO epochs	3
Number of environments	16
Steps per environment	512
Minibatches per epoch	8
Total timesteps	6M
Framework	Custom PPO

Table 16: **Hyperparameters for PPO-LSTM.**

Hyperparameter	Value
LSTM hidden size	512
Actor network	[128, 128]
Critic network	[128, 128]
Learning rate (initial)	3×10^{-4}
Learning rate (final)	1×10^{-5}
LR schedule	Linear
Discount (γ)	0.99
GAE lambda (λ)	0.98
Gradient clipping	0.5
Entropy coefficient	0.01
Value loss coefficient	0.5
Sequence length	128
Minibatch size	512
PPO epochs	10
Number of environments	16
Steps per environment	128
Total timesteps	6M
Framework	JAX
Parameters	≈2.3M

Table 17: **DNC architecture hyperparameters (POPGym implementation).**

Hyperparameter	Value
Controller (LSTM) hidden size	64
Memory slots	16
Word size	16
Write heads	1
Read heads	4
Observation feature size	64
Clip value	20.0
Total timesteps	6M
Framework	POPGym

Table 18: **Hyperparameters for Resettable S5.** Tuples list Endless T-Maze, Rune T-Maze, and Rune Multi-Corridor.

Hyperparameter	Value
S5 layers	$(2, 2, 4)$
Model width	256
State size	$(256, 128, 256)$
Blocks per layer	1
Activation	Full GLU
Normalization	None
Learning rate	$(5, 30, 30) \times 10^{-5}$
LR schedule	Constant
Discount (γ)	0.99
GAE lambda (λ)	1.0
Entropy coefficient	0.0
PPO clipping	0.2
Value loss coefficient	1.0
Gradient clipping	0.5
PPO epochs	30
Number of environments	32
Steps per environment	1024
Minibatch size	8192
Total timesteps	6M
Framework	JAX

G COMPUTE AND LLM USAGE

The vector-environment agents other than DNC were trained on NVIDIA H100 GPUs. ALER performs one read and one write over $C = 10$ slots per step, in addition to the LSTM update. A run of 2M environment steps takes 24.9 minutes on average over 613 logged runs. DNC, in the POPGym implementation, was trained on CPUs, with 462 to 470 minutes per run and 742 CPU-hours in total. All GPU-based experiments reported in this work were conducted using 1 NVIDIA H100 GPU. Large language models were used for polishing and improving the clarity of the paper text.