ELMUR: External Layer Memory with Update/Rewrite for Long-Horizon RL

Anonymous Author(s)
Affiliation
Address
email

Abstract: Real-world robotic agents must act under partial observability and long horizons, where key cues may appear long before they affect decision making. Standard recurrent or transformer models struggle: context windows truncate history, and naive memory extensions fail under scale and sparsity. We propose **ELMUR** (**E**xternal **L**ayer **M**emory with **U**pdate/**R**ewrite), a transformer architecture with structured external memory. Each layer maintains memory embeddings, interacts with them via bidirectional cross-attention, and updates them through an Least Recently Used **(LRU)** memory module using replacement or convex blending. This design extends effective horizons up to 100,000 times beyond the context length. On synthetic benchmarks and robotic manipulation tasks from MIKASA-Robo with sparse rewards, ELMUR consistently outperforms other strong baselines, achieving robust long-term recall and showing promising results where prior models struggle. Our results show that structured external memory is a simple and effective recipe for scalable decision making under partial observability.

Keywords: Memory, Robotics, POMDP

1 Introduction

Imagine a robot preparing a meal: it chops vegetables, adds spices, and simmers a sauce, only to forget it already added salt. This illustrates partial observability — the world rarely provides all necessary information. Humans overcome this with seamless recall, a skill robotics urgently needs.

Robotics is at a crossroads. While robots excel in controlled environments, their competence often collapses over longer horizons. They forget fleeting but essential clues — a tool placed aside, an ingredient already added. Recurrent or transformer models with limited memory fail under the scale and sparsity of real-world tasks. The most promising direction lies in hybrid architectures, combining transformers with persistent memory.

Reinforcement Learning (RL) [1] is the standard paradigm, but real-world trial-and-error demands huge interaction budgets and raises safety concerns. Simulation is cheaper, but the sim-to-real gap remains. Offline RL learns from pre-collected data, avoiding active interaction, yet both online and offline RL usually assume dense step-by-step rewards. In practice, rewards are sparse: often only a binary success signal is available.

Imitation Learning (IL) addresses this by learning directly from expert demonstrations. Its simplest form, Behavior Cloning (BC), reduces control to supervised learning. Recent Vision-Language-Action (VLA) models map images and instructions to actions from large datasets. However, transformers in VLAs are restricted by a fixed context window, which causes three fundamental challenges: (i) extending context length without quadratic costs, (ii) mitigating forgetting from truncated windows, and (iii) supporting decision making under partial observability by retaining task-relevant information over long horizons. This motivates our central question: *how can we equip IL policies with efficient long-term memory to solve long-horizon, partially observable tasks?*

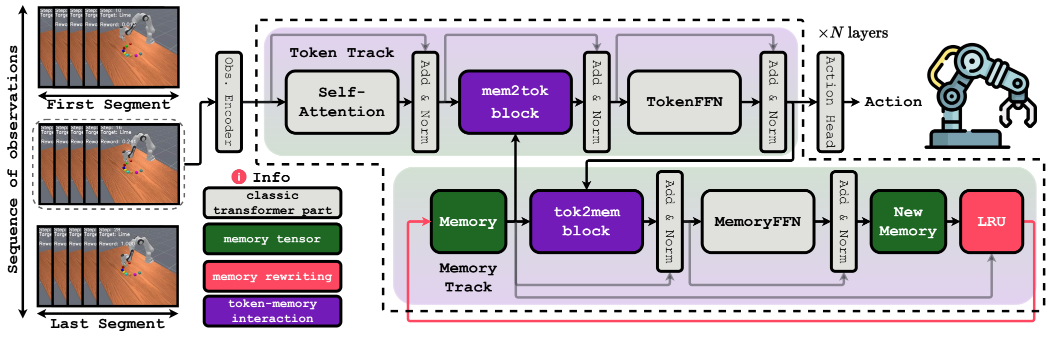

Figure 1: **ELMUR overview.** Each transformer layer is augmented with an external memory track that runs in parallel with the token track. Tokens attend to memory through a `mem2tok` block, while memory slots are updated from tokens through a `tok2mem` block. A Least Recently Used (LRU) policy selectively rewrites memory slots via full replacement or convex blending, ensuring bounded yet persistent storage. This design enables precise token↔memory interaction and long-horizon recall beyond the native attention window.

To address this challenge, we introduce **ELMUR** (**E**xternal **L**ayer **M**emory with **U**pdate/**R**ewrite), a transformer architecture where each layer is equipped with a structured external memory (Figure 1). ELMUR integrates three key ideas: (i) **layer-local memory embeddings** that persist across segments, (ii) cross-attention **blocks for token–memory interaction** (`tok2mem`, `mem2tok`), and (iii) an **LRU memory module** that rewrites embeddings via replacement or convex blending, balancing plasticity and stability. This design extends recall up to $100{,}000\times$ beyond the attention window, enabling efficient long-horizon reasoning in robotics, where long contexts are infeasible.

We evaluate ELMUR on synthetic and robotic benchmarks designed to test memory, including visual tabletop manipulation with sparse binary rewards. In all cases, ELMUR surpasses strong baselines, achieving state-of-the-art results in long-horizon reasoning, selective memory, and generalization under partial observability.

Our contributions:

- **We propose ELMUR**, a transformer architecture with external layer-local memory and bidirectional token↔memory cross-attention – Section 4.
- **We design an LRU-based update rule** that rewrites memory through replacement and convex blending, ensuring bounded yet persistent storage – Section 4.6.
- **We show empirically that ELMUR scales memory capacity** up to $100{,}000\times$ the attention length and achieves state-of-the-art performance on synthetic and robotic benchmarks, including sparse-reward tabletop manipulation – Section 5.

2 Related Work

Memory in Deep Learning. Research on memory in deep learning explicit stores and implicit mechanisms. Explicit stores provide read–write access and structured retrieval [2, 3, 4, 5, 6]. Implicit mechanisms compress history into hidden states or dynamical systems, as in LSTMs and their variants [7, 8], linear-attention Transformers [9], RWKV [10], and state-space families [11, 12].

Beyond the fixed self-attention window of the Transformer [13], many extensions have been proposed to mitigate the limitation of finite context. A broad family of approaches extends effective horizons using caches, compression, or external memory modules [14, 15, 16, 17, 18, 19, 20, 21, 22]. These methods differ in how they manage past information—whether through segment-level recurrence, hierarchical compression, or associative retrieval—but all aim to overcome the hard cutoff imposed by standard transformers. Recent work on test-time memorization [23] further explores adaptive long-term memory learned dynamically during inference.

Memory in Reinforcement Learning. In RL partial observability makes memory essential. One line of work develops spatial and episodic buffers such as Neural Map [24], Hierarchical Chunk Attention Memory [25], and Stable Hadamard Memory [26], alongside taxonomies for evaluation [27]. Another line adapts sequence models for control: Gated TrXL [28], Approximate Gated Linear Attention (AGaLiTe) [29] scales long horizons, and AMAGO-2 [30] demonstrates meta-RL capabilities. Finally, explicit external memory has been added to transformers: the Recurrent Action Transformer with Memory (RATE) [31] preserves information across segments, while Fast-and-Forgetful Memory [32] applies controlled decay and Re:Frame [33] retrieves relevant subsequences from an associative index.

We propose the ELMUR model: a transformer where each layer is paired with a structured external memory and explicit read–write operations. Unlike methods that simply cache past states or apply lossy compression, ELMUR maintains layer-local memory embeddings, updates them with an LRU-based mechanism balancing stability and plasticity, and grounds access in relative temporal biases. This design enables robust long-horizon reasoning under partial observability, while avoiding the inefficiency and brittleness of prior memory extensions.

3 Background

Partially Observable Markov Decision Processes (POMDPs). Many real-world robotic and control tasks involve partial observability, where the agent does not have direct access to the system state [34]. This setting is modeled as a partially observable Markov decision process (POMDP), defined as the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, \rho_0, \gamma)$, with latent state space $\mathcal{S}$, action space $\mathcal{A}$, and observation space $\mathcal{O}$. The transition dynamics are defined as $T : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, where $T(s' \mid s, a)$ denotes the probability of reaching state s' when action a is applied in state s. The observation function is $Z : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{O})$, where $Z(o \mid s', a)$ denotes the likelihood of observing o after reaching state s' under action a. The reward function is $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, the initial state distribution is $\rho_0 \in \Delta(\mathcal{S})$, and $\gamma \in (0, 1)$ is the discount factor.

In fully observable MDPs, the optimal policy depends only on the current state, $\pi^*(a_t \mid s_t)$. In POMDPs, however, the agent cannot access s_t directly. Instead, the optimal policy must condition on the full history $h_t = (o_0, a_0, o_1, a_1, \ldots, o_t)$, yielding $\pi^*(a_t \mid h_t)$. A sufficient statistic of h_t is the *belief state* $b_t \in \Delta(\mathcal{S})$, defined as $b_t(s) = \Pr(s_t = s \mid h_t)$. Exact belief updates are typically infeasible in high-dimensional domains such as robotics.

A practical alternative is to approximate the history with a learned *memory state* $m_t = f_\phi(m_{t-1}, o_t, a_{t-1})$, $\pi_\theta : \mathcal{M} \to \Delta(\mathcal{A})$, $\pi_\theta(a_t \mid m_t)$, where f_ϕ is a recurrent or memory-augmented update rule (e.g., RNNs, transformers with external memory). In the fully observable case, the memory state reduces to the true state, $m_t \equiv s_t$, and the POMDP reduces to a standard MDP.

4 Method

Many real-world decision-making tasks involve long horizons and partial observability, where key information may appear thousands of steps before it is needed. Standard transformers are constrained by a fixed attention window: longer contexts incur quadratic cost, while truncation causes forgetting. Recurrent models help but suffer from vanishing information and instability. Efficient long-term reasoning thus requires an explicit mechanism to store and retrieve task-relevant information across very long trajectories.

To this end, we propose **ELMUR** (External Layer Memory with Update/Rewrite), a GPT-style transformer decoder augmented with structured external memory. Unlike architectures that simply cache hidden states, ELMUR equips each layer with its own memory track and explicit read–write operations, enabling persistent storage and selective updating. This design directly addresses the challenges of long-horizon reinforcement learning under partial observability, where dependencies extend far beyond the context length of standard transformers.

Algorithm 1 ELMUR layer (per segment i, per layer ℓ)

```
Require: Token states $h \in \mathbb{R}^{B\times L\times d}$, memory state $(m,p)$ with memory tensor $m \in \mathbb{R}^{B\times M\times d}$ and
    memory anchors $p \in \mathbb{Z}^{B\times M}$, absolute token times $t \in \mathbb{Z}^{0+}$, where $T$ - context length; $M$ -
    number of memory slots; $B$ - batch size; $d$ - model dimension.
Ensure: Updated token states $h'$, updated memory state $(m',p')$
 1: // Input embedding (before first layer)
 2: $h \leftarrow \mathrm{ObsEncoder}(o)$    ▷ project observations $o$ into hidden state $h$
 3: // Token track (per layer)
 4: $h \leftarrow \mathrm{AddNorm}\big(h + \mathrm{SelfAttention}(h; \text{causal\_mask})\big)$
 5: $B_{\mathrm{read}} \leftarrow \mathrm{RelativeBias}(t,p)$    ▷ read bias, tokens→memory
 6: $h \leftarrow \mathrm{AddNorm}\big(h + \mathrm{CrossAttention}(Q{=}h, K{=}m, V{=}m;\ \mathrm{non\_causal},\ B_{\mathrm{read}})\big)$
 7: $h \leftarrow \mathrm{AddNorm}\big(h + \mathrm{TokenFFN}(h)\big)$
 8: $h' \leftarrow h$
 9: // Output decoding (after final layer)
10: $a \leftarrow \mathrm{ActionHead}(h')$    ▷ map final hidden state $h'$ into action distribution
11: // Memory track
12: $B_{\mathrm{write}} \leftarrow -B_{\mathrm{read}}$    ▷ reversed relative bias
13: $u \leftarrow \mathrm{AddNorm}\big(m + \mathrm{CrossAttention}(Q{=}m, K{=}h', V{=}h';\ \mathrm{non\_causal\_mask},\ B_{\mathrm{write}})\big)$
14: $\tilde{u} \leftarrow \mathrm{AddNorm}\big(u + \mathrm{MemoryFFN}(u)\big)$
15: $(m',p') \leftarrow \mathrm{LRU\_Update}(m, p, \tilde{u},\ t_0{=}t_1)$    ▷ $t_1$: newest absolute time in segment $i$
16: return $(h',\ (m',p'))$
```

4.1 ELMUR Overview

As shown in Figure 1, ELMUR is organized into two tracks: the **token track**, which encodes observations and produces actions, and the **memory track**, which maintains a memory store and processing across segments. This two tracks interact bidirectionally: tokens read from memory through a cross-attention block and, symmetrically, write their representations back through another. Both operations are guided by special relative biases computed from the absolute timestep of each token and the anchor of the corresponding memory slot, defined as the last timestep when that slot was updated. When writing, new entries fill available slots; if none are free, the least recently used slots are refreshed via convex combinations of old and new content, providing controlled forgetting. Memory is updated recurrently across trajectory segments: each segment is processed in order, and its final hidden states update the memory before passing it forward. In this way, ELMUR builds and maintains temporally grounded memory that supports long-horizon reasoning. The following subsections describe each stage of ELMUR, and the complete procedure is summarized in Algorithm 1.

Algorithm 2 LRU Update

```
Require: Current memory $(m,p)$ (may be uninitialized), candidate update $\tilde{u}$, newest
    anchor time $t_0$, blend $\lambda \in [0,1]$, init scale $\sigma$
Ensure: Updated memory $(m',p')$
 1: (Initialization) if $m,p$ uninitialized then
 2:     $m \leftarrow \mathcal{N}(0,\sigma^2 I)$    ▷ initial slots
 3:     $p \leftarrow -1$ array    ▷ sentinel anchors
 4: end if
 5: empty $\leftarrow (p < 0)$
 6: if any empty then
 7:     write_mask $\leftarrow$ one_hot(first empty)
 8:     $\alpha \leftarrow 1.0$
 9: else
10:     $j^\star \leftarrow \arg\min p$    ▷ oldest anchor
11:     write_mask $\leftarrow$ one_hot($j^\star$)
12:     $\alpha \leftarrow \lambda$
13: end if
14: blend $\leftarrow \alpha\tilde{u} + (1-\alpha)m$
15: $m' \leftarrow$ select(write_mask, blend, $m$)
16: $p' \leftarrow$ select(write_mask, $t_0$, $p$)
17: return $(m',p')$
```

4.2 Segment-Level Recurrence

Long trajectories cannot be fed to a transformer in full: self-attention scales quadratically with sequence length. Splitting trajectories into shorter segments reduces the context size, but creates a new problem: how to transmit information across segment boundaries without large caches or lossy compression. Segment-level recurrence addresses this by making the transformer operate as an RNN over segments, sequentially processing segments while carrying a memory state forward [14, 17].

We adopt this paradigm but realize the carried memory state as layer-local external memory rather than cached hidden activations. Each layer maintains a persistent memory that (i) is read by tokens of the current segment and (ii) is updated from those tokens before moving to the next segment. This yields stable training (no backprop through the entire trajectory), low memory use, and minimal information break between segments.

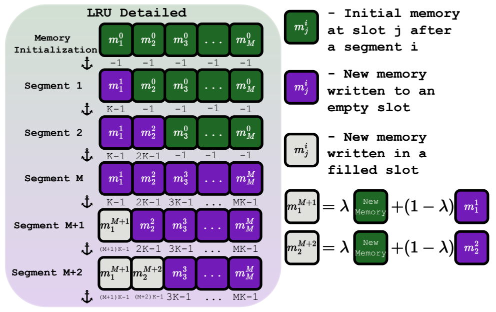
LRU Detailed
Memory Initialization
Segment 1
Segment 2
Segment M
Segment M+1
Segment M+2
m_1^0 m_2^0 m_3^0 ... m_M^0
-1 -1 -1 -1 -1
K-1 2K-1 3K-1 ... MK-1
(M+1)K-1 (M+2)K-1
- Initial memory at slot j after a segment i
- New memory written to an empty slot
- New memory written in a filled slot
$m_1^{M+1} = \lambda$ New Memory $+(1-\lambda)$ m_1^1
$m_2^{M+2} = \lambda$ New Memory $+(1-\lambda)$ m_2^2

Figure 2: **LRU-based memory management in ELMUR.** Each layer maintains M memory slots, initialized with random vectors (green). As new segments arrive, tokens write updates into empty slots (purple) by full replacement. Once all slots are filled, the least recently used slot is refreshed via a convex update that blends new content with the previous memory. Anchors below each row indicate the timestep of the most recent update. This scheme ensures bounded capacity while preserving long-horizon information.

Let a trajectory of length T be $\mathbf{o}_{1:T} = (o_1, \ldots, o_T)$ and fix context length L. Partition into $S = \lceil T/L \rceil$ contiguous segments $\mathcal{S}_i = (o_{(i-1)L+1}, \ldots, o_{\min(iL,T)})$, $i = 1, \ldots, S$. Each layer holds memory slots $\mathbf{m}$, updated at the end of segment $i-1$ and reused at segment i (see Figure 1). The token track for segment i attends to the detached previous memory, $\mathbf{h}^{(i)} = \mathrm{TokenTrack}\big(\mathcal{S}_i,\ \mathrm{sg}(\mathbf{m}^{\,i-1})\big)$,, where $\mathrm{sg}(\cdot)$ stops gradients across segments. After processing $\mathcal{S}_i$, the memory is rewritten from $\mathbf{h}^{(i)}$ via the write path (tok2mem) and the LRU update rule (Section 4.6), yielding $\mathbf{m}^{\,i}$ that is passed to the next segment. Thus, ELMUR implements segment-level recurrence with explicit, structured memory instead of hidden-state caches.

4.3 Token Track

The token track in ELMUR autoregressively models dependencies within the current context and integrates them with external memory. Standard transformers rely only on self-attention inside a fixed window, which limits access to distant information. In ELMUR, the token track augments this local modeling by retrieving relevant content from memory, so that predictions can depend on both recent context and long-past events.

Within each segment, the token track first encodes local dependencies through self-attention and then enriches these representations by reading from the memory via the `mem2tok` block. In this way, tokens can directly access task-relevant information stored many segments earlier, overcoming the cutoff imposed by the finite context window.

Encoded observations are first processed with self-attention using relative positional encodings [14] and a causal mask: $\mathbf{h}_{\text{sa}} = \mathrm{AddNorm}(\mathbf{x} + \mathrm{SelfAttention}(\mathbf{x}))$. Unlike Transformer-XL, ELMUR does not cache hidden states across segments; long-term dependencies are managed entirely by the external memory. The hidden states then enter the `mem2tok` block, a cross-attention module where tokens act as queries and memory slots provide keys and values:

$$\mathbf{h}_{\text{mem2tok}} = \mathrm{AddNorm}(\mathbf{h}_{\text{sa}} + \mathrm{CrossAttention}(Q = \mathbf{h}_{\text{sa}},\ K,\ V = \mathbf{m}))\,. \tag{1}$$

This block uses a non-causal mask, allowing tokens unrestricted access to memory, and applies a relative bias reflecting the temporal distance between tokens and memory anchors (see subsection 4.5). Finally, outputs are refined by a feed-forward network with residual connection and normalization:

$$\mathbf{h} = \mathrm{AddNorm}(\mathbf{h}_{\text{mem2tok}} + \mathrm{FFN}(\mathbf{h}_{\text{mem2tok}}))\,, \tag{2}$$

and the final-layer states are passed to an action head to produce the predicted action.

4.4 Memory Track

While the token track enables tokens to read from memory, long-horizon reasoning also requires writing new information back. Without an explicit write mechanism, the model would either forget past events or accumulate unstructured caches that quickly become inefficient. The memory track addresses this by providing a controlled way for tokens to update persistent memory, ensuring that salient information is retained across segments while older or less useful content can be overwritten.

Each transformer layer in ELMUR maintains its own memory matrix. After processing a segment, tokens contribute updates to this memory through the `tok2mem` block. The resulting updates are refined and then integrated using an LRU module, which guarantees bounded yet persistent storage.

Formally, each layer stores memory slots $\mathbf{m} \in \mathbb{R}^{M\times d}$. Token representations are passed into the `tok2mem` block, a cross-attention module where memory slots act as queries and token states as keys and values: $\mathbf{m}_{\text{tok2mem}} = \mathrm{AddNorm}(\mathbf{m} + \mathrm{CrossAttention}(Q = \mathbf{m},\, K, V = \mathbf{h}))$. As in `mem2tok`, a non-causal mask is used, but the relative bias is reversed, encouraging tokens to update slots anchored near their own timestep. The updates are then refined by a feed-forward network with residual connection and normalization: $\mathbf{m}_{\text{new}} = \mathrm{AddNorm}(\mathbf{m}_{\text{tok2mem}} + \mathrm{FFN}(\mathbf{m}_{\text{tok2mem}}))$.

Finally, $\mathbf{m}_{\text{new}}$ is integrated with existing slots via the LRU mechanism (see Figure 2, Algorithm 2), which fills free slots or refreshes the least recently used ones by convex blending. This ensures that memory remains bounded yet persistently updated with the most relevant information.

4.5 Relative Bias in Cross-Attention

When memory spans multiple segments, absolute positions alone are insufficient: the same token index may correspond to very different points in the overall trajectory. Without a proper temporal signal, the model cannot reliably decide which memory slots to read from or update. To maintain consistency across arbitrarily long sequences, we need a mechanism that encodes relative distances between tokens and memory entries.

ELMUR introduces a learned relative bias that conditions cross-attention scores on the temporal offset between a token's timestep and the anchor of a memory slot (the last time the slot was updated). This allows the model to favor temporally close interactions while still permitting access to distant information. Concretely, we add a relative bias $\mathbf{B}_{\text{rel}}$ to the cross-attention logits: $\mathrm{Attn}(\mathbf{Q}, \mathbf{K}) = \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_h}} + \mathbf{B}_{\text{rel}}$. In the `mem2tok` path (reading), the bias emphasizes nearby slots, promoting retrieval from recent memory while keeping older slots accessible. In the `tok2mem` path (writing), the bias is inverted, directing updates toward slots aligned with the writing tokens. By grounding access in relative rather than absolute time, ELMUR generalizes beyond the fixed context window and maintains temporally coherent memory interactions across long horizons (see Figure 1).

4.6 Memory Management with LRU

External memory must remain bounded: storing every token across long trajectories is infeasible. At the same time, simply discarding old information risks catastrophic forgetting. A principled mechanism is therefore needed to decide which slots to update, overwrite, or preserve as new content arrives.

ELMUR introduces a Least Recently Used (LRU) block (Figure 2, Algorithm 2) that manages a fixed set of M slots in each layer. Each slot stores both a vector and an anchor indicating the timestep of its most recent update. By always refreshing the least recently used slot, the block guarantees bounded capacity while maintaining relevant long-term context.

At training start, **initialization** samples memory embeddings from $\mathcal{N}(0, \sigma^2 I)$ and marks them empty, making them the first to be overwritten. While empty memory embeddings remain, **full replacement** inserts new memory vectors directly, ensuring fresh content without blending. Once all memory embeddings are filled, the block switches to **convex update**, where the least recently

Table 1: Comparison of success rates (mean $\pm$ standard error) on MIKASA-Robo tasks. Results are averaged over 3 runs and validated on 100 seeds. ELMUR consistently outperforms baselines, demonstrating stronger memory in robotic manipulation. For transformer models (RATE, DT, ELMUR), context length was set to one third of the maximum episode length.

Task	**RATE**	**DT**	**BC-MLP**	**CQL-MLP**	**DP**	**ELMUR (ours)**
RememberColor3-v0	0.65±0.04	0.01±0.01	0.27±0.03	0.29±0.01	0.32±0.01	**0.89±0.07**
RememberColor5-v0	0.13±0.03	0.07±0.05	0.12±0.01	0.15±0.02	0.10±0.02	**0.19±0.03**
RememberColor9-v0	0.09±0.02	0.01±0.01	0.12±0.02	0.15±0.01	0.17±0.01	**0.23±0.02**
TakeItBack-v0	0.42±0.24	0.08±0.04	0.33±0.10	0.04±0.01	0.05±0.02	**0.78±0.03**

used memory embedding (oldest anchor) is updated by

$$\mathbf{m}_j^{i+1} = \lambda\,\mathbf{m}_{\text{new}} + (1-\lambda)\,\mathbf{m}_j^{i}, \tag{3}$$

with $\lambda \in [0,1]$ controlling the trade-off between overwriting and retention.

Through this policy, ELMUR fully utilizes capacity before overwriting and applies gradual blending when replacement becomes necessary, preserving long-horizon information on arbitrarily long trajectories while keeping memory strictly bounded.

By integrating token-level processing with an explicit memory system, ELMUR achieves three properties: (i) temporally grounded read–write access via relative-bias cross-attention, (ii) bounded yet flexible capacity through the LRU-based memory manager, and (iii) scalable long-horizon learning enabled by segment-level recurrence.

5 Experiments

We evaluate ELMUR on synthetic [35] and robotic benchmarks [36] designed to probe long-horizon memory under partial observability. Our goals are threefold: (i) to test whether ELMUR retains information across horizons far beyond its context window, (ii) to assess generalization across sequence lengths, and (iii) to demonstrate effectiveness in robotic manipulation tasks with sparse rewards. We compare against strong baselines, including transformer-based models: Decision Transformer (DT) [37] and Recurrent Action Transformer with Memory (RATE) [31]; state-space models (DMamba) [38]; and imitation or offline RL methods: Behavior Cloning (BC), Conservative Q-Learning (CQL) [39], and Diffusion Policy (DP) [40].

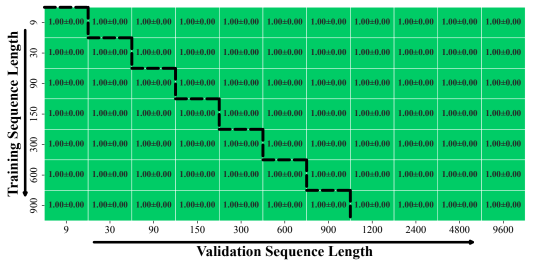

Figure 3: Generalization of ELMUR across T-Maze sequence lengths. Each cell shows success rate (mean $\pm$ standard error) for a given pair of training and validation sequence lengths. Results demonstrate perfect transfer: models trained on shorter sequences retain 100% performance when evaluated on sequences up to 9600 steps. Each training length was divided into three segments of equal length during training.

5.1 Memory-intensive environments

Passive T-Maze. The *Passive T-Maze* [35] features a corridor ending in a junction with two goals. At the start, one goal is randomly revealed, and the agent must recall this cue after traversing the corridor to choose the correct branch. Observations are vectors; actions are discrete. Rewards are sparse, provided only upon reaching the correct goal. The task tests whether the model can retain early cues across long delays.

MIKASA-Robo benchmark. We further evaluate on the MIKASA-Robo benchmark [36], which provides robotic tabletop manipulation tasks designed for memory evaluation. Each environment simulates a 7-DoF arm with a two-finger gripper. Observations are paired RGB images ($3 \times 128 \times 128$) from a static and wrist camera; actions are continuous (7 joints + gripper). Rewards are binary, given only on task success. We study two families of MIKASA-Robo tasks: (i) `RememberColor[3,5,9]-v0`, where the agent must recall the color of a hidden cube after a delay with distractors, and (ii) `TakeItBack-v0`, where the agent first moves a cube to a goal, then must return it once the goal changes.

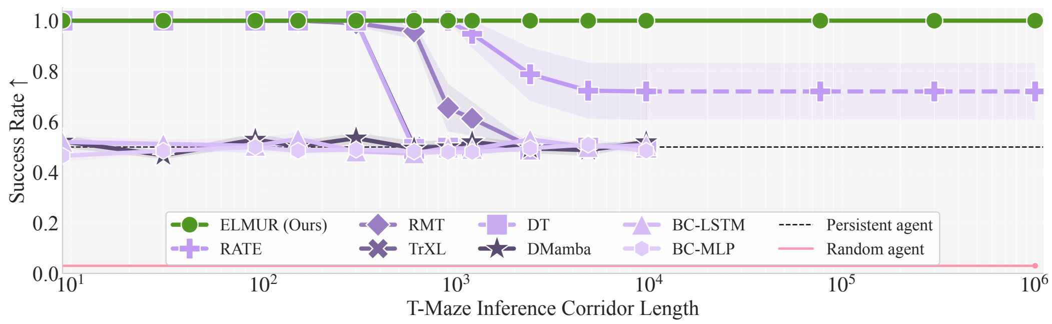

Figure 4: Success rate on the Passive T-Maze task as a function of inference corridor length. Shaded areas denote mean $\pm$ standard error of the mean over 4 independent runs, each evaluated on 100 random seeds. **ELMUR achieves a 100% success rate up to corridor lengths of one million steps.** In this figure, the context length is $L = 10$ with $S = 3$ segments; thus **ELMUR carries information across horizons 100,000 times longer than its context window**.

5.2 Discussion of Results

Our experiments highlight ELMUR's ability to handle long-horizon dependencies under partial observability.

T-Maze. ELMUR achieves 100% success on corridors up to one million steps (Figure 4), despite a context of only $L = 10$ tokens and $S = 3$ segments during training. Competing transformers (DT, TrXL, RMT, RATE) fail once horizons exceed their context, and state-space models (DMamba) collapse entirely. This shows that segment-level recurrence with layer-local memory enables qualitatively stronger generalization, solving the memory bottleneck rather than just extending the cutoff. As shown in Figure 3, ELMUR trained on short contexts (3 or 30 steps) transfers perfectly to sequences up to 9600, maintaining 100% success. This demonstrates robust scaling and the ability to extrapolate/interpolate beyond training lengths.

MIKASA-Robo. On robotic manipulation (Table 1), ELMUR outperforms DT, RATE, DP, and offline RL methods (CQL, BC). Gains are largest in temporally demanding tasks: in `TakeItBack-v0`, ELMUR nearly doubles the next-best success rate, while in the `RememberColor` family it remains stable as distractors increase, unlike baselines. Selective rewriting via the LRU module proves critical for balancing stability and plasticity.

These results show that ELMUR's design — external memory, relative-bias cross-attention, and LRU updates — yields robust long-horizon reasoning, effective decision making under sparse rewards, and scalable memory for robotics.

6 Limitations

Despite strong long-horizon performance, ELMUR has several limitations. First, memory management relies on a simple LRU policy with fixed blending, which may overwrite useful content if memory size, segment length, or blend rate are poorly chosen. Second, cross-attention between tokens and memory adds cost proportional to memory size per layer; while cheaper than extending the token window, it still increases inference time for larger models. Third, evaluation is limited to synthetic and simulated tabletop tasks under imitation learning with sparse rewards; online RL stability, safety, and real-robot trials are not explored.

7 Conclusion

We introduced ELMUR, a transformer with layer-local external memory, relative-bias cross-attention for temporally grounded read/write, and an LRU-based update that balances stability and plasticity. ELMUR extends horizons far beyond the attention window, processing sequences up to 100,000 times longer than its context, and achieves state-of-the-art results on MIKASA-Robo, consistently surpassing strong baselines. Looking forward, we identify three directions: learning the memory controller (adaptive allocation, content-aware eviction, sparse writes) instead of fixed LRU, and developing hierarchical or multimodal memory to integrate visual, language, and proprioceptive cues. We hope ELMUR provides a practical recipe for transformer policies with scalable memory in real-world sequential decision making.

References

[1] R. S. Sutton, A. G. Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[2] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[3] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwinska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. P. Agapiou, A. P. Badia, K. M. Hermann, Y. Zwols, G. Ostrovski, A. Cain, H. King, C. Summerfield, P. Blunsom, K. Kavukcuoglu, and D. Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538: 471–476, 2016. URL `https://api.semanticscholar.org/CorpusID:205251479`.

[4] J. Weston, S. Chopra, and A. Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[5] A. Santoro, S. Bartunov, M. M. Botvinick, D. Wierstra, and T. P. Lillicrap. Meta-learning with memory-augmented neural networks. In *International Conference on Machine Learning*, 2016. URL `https://api.semanticscholar.org/CorpusID:6466088`.

[6] A. H. K. Ahmadi. *Memory-based graph networks*. University of Toronto (Canada), 2020.

[7] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997. URL `https://api.semanticscholar.org/CorpusID:1915014`.

[8] M. Beck, K. Pöppel, M. Spanring, A. Auer, O. Prudnikova, M. Kopp, G. Klambauer, J. Brandstetter, and S. Hochreiter. xlstm: Extended long short-term memory. *Advances in Neural Information Processing Systems*, 37:107547–107603, 2024.

[9] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

[10] B. Peng, E. Alcaide, Q. Anthony, A. Albalak, S. Arcadinho, S. Biderman, H. Cao, X. Cheng, M. Chung, M. Grella, et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

[11] A. Gu, K. Goel, and C. Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

[12] A. Gu and T. Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[14] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[15] J. W. Rae, A. Potapenko, S. M. Jayakumar, and T. P. Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

[16] Y. Wu, M. N. Rabe, D. Hutchins, and C. Szegedy. Memorizing transformers. *arXiv preprint arXiv:2203.08913*, 2022.

[17] A. Bulatov, Y. Kuratov, and M. Burtsev. Recurrent memory transformer. *Advances in Neural Information Processing Systems*, 35:11079–11091, 2022.

[18] H. Liu, M. Zaharia, and P. Abbeel. Ring attention with blockwise transformers for near-infinite context. *arXiv preprint arXiv:2310.01889*, 2023.

[19] M. S. Burtsev, Y. Kuratov, A. Peganov, and G. V. Sapunov. Memory transformer. *arXiv preprint arXiv:2006.11527*, 2020.

[20] Q. Wu, Z. Lan, K. Qian, J. Gu, A. Geramifard, and Z. Yu. Memformer: A memory-augmented transformer for sequence modeling. *arXiv preprint arXiv:2010.06891*, 2020.

[21] I. Rodkin, Y. Kuratov, A. Bulatov, and M. Burtsev. Associative recurrent memory transformer. *arXiv preprint arXiv:2407.04841*, 2024.

[22] S. Ding, J. Shang, S. Wang, Y. Sun, H. Tian, H. Wu, and H. Wang. Ernie-doc: A retrospective long-document modeling transformer. *arXiv preprint arXiv:2012.15688*, 2020.

[23] A. Behrouz, P. Zhong, and V. Mirrokni. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*, 2024.

[24] E. Parisotto and R. Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *arXiv preprint arXiv:1702.08360*, 2017.

[25] A. Lampinen, S. Chan, A. Banino, and F. Hill. Towards mental time travel: a hierarchical memory for reinforcement learning agents. *Advances in Neural Information Processing Systems*, 34:28182–28195, 2021.

[26] H. Le, K. Do, D. Nguyen, S. Gupta, and S. Venkatesh. Stable hadamard memory: Revitalizing memory-augmented agents for reinforcement learning. *arXiv preprint arXiv:2410.10132*, 2024.

[27] E. Cherepanov, N. Kachaev, A. Zholus, A. K. Kovalev, and A. I. Panov. Unraveling the complexity of memory in rl agents: an approach for classification and evaluation. *arXiv preprint arXiv:2412.06531*, 2024.

[28] E. Parisotto, F. Song, J. Rae, R. Pascanu, C. Gulcehre, S. Jayakumar, M. Jaderberg, R. L. Kaufman, A. Clark, S. Noury, et al. Stabilizing transformers for reinforcement learning. In *International conference on machine learning*, pages 7487–7498. PMLR, 2020.

[29] S. Pramanik, E. Elelimy, M. C. Machado, and A. White. Agalite: Approximate gated linear transformers for online reinforcement learning. *arXiv preprint arXiv:2310.15719*, 2023.

[30] J. Grigsby, J. Sasek, S. Parajuli, D. Adebi, A. Zhang, and Y. Zhu. Amago-2: Breaking the multi-task barrier in meta-reinforcement learning with transformers. *Advances in Neural Information Processing Systems*, 37:87473–87508, 2024.

[31] E. Cherepanov, A. Staroverov, D. Yudin, A. K. Kovalev, and A. I. Panov. Recurrent action transformer with memory. *arXiv preprint arXiv:2306.09459*, 2023.

[32] S. Morad, R. Kortvelesy, S. Liwicki, and A. Prorok. Reinforcement learning with fast and forgetful memory. *Advances in Neural Information Processing Systems*, 36:72008–72029, 2023.

[33] D. Zelezetsky, E. Cherepanov, A. K. Kovalev, and A. I. Panov. Re: Frame–retrieving experience from associative memory. *arXiv preprint arXiv:2508.19344*, 2025.

[34] M. Lauri, D. Hsu, and J. Pajarinen. Partially observable markov decision processes in robotics: A survey. *IEEE Transactions on Robotics*, 39(1):21–40, 2022.

[35] T. Ni, M. Ma, B. Eysenbach, and P.-L. Bacon. When do transformers shine in rl? decoupling memory from credit assignment. *Advances in Neural Information Processing Systems*, 36: 50429–50452, 2023.

[36] E. Cherepanov, N. Kachaev, A. K. Kovalev, and A. I. Panov. Memory, benchmark & robots: A benchmark for solving complex tasks with reinforcement learning. *arXiv preprint arXiv:2502.10550*, 2025.

[37] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[38] T. Ota. Decision mamba: Reinforcement learning via sequence modeling with selective state spaces. *arXiv preprint arXiv:2403.19925*, 2024.

[39] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *Advances in neural information processing systems*, 33:1179–1191, 2020.

[40] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.