

Does Learning Recurrent Networks Matter in Partially Observable RL?

Zekang Wang
University of Macau
Macau SAR, China
yc67543@um.edu.mo

Zhe He
University of Macau
Macau SAR, China
yc67202@um.edu.mo

Gaojie Jin*
University of Macau
Macau SAR, China
gaojiejin@um.edu.mo

Abstract

Reinforcement learning (RL) under partial observability typically relies on recurrent networks to encode temporal information, with the recurrent module trained jointly with the policy or value function. This raises a natural question: is learning the recurrent network actually necessary? In this paper, we investigate whether a frozen recurrent network can provide sufficient memory for solving partially observable Markov decision processes (POMDPs). Motivated by the theoretical foundations of Reservoir Computing (RC) and Generalized Reservoir Computing (GRC), we study a simple architecture that combines a fixed recurrent network with a trainable nonlinear output module. We evaluate this approach across tasks with varying memory demands, observation modalities, and action spaces. Surprisingly, frozen recurrent networks support recall over hundreds of steps and achieve competitive control performance across diverse POMDPs, while using less than half the number of trainable parameters required by their fully trainable counterparts. These results suggest that learning the recurrent memory module is not always necessary for solving POMDPs. Frozen recurrent networks therefore provide a simple, parameter-efficient, and surprisingly strong baseline for partially observable RL. Our source code is available at <https://github.com/Aphlios/FRN-POMDP>.

Keywords

Reinforcement Learning, POMDP, Recurrent Neural Network, Reservoir Computing

1 Introduction

Reinforcement learning (RL) is commonly formulated as a Markov Decision Process (MDP), in which the current state contains all information needed for optimal decision making [56]. In many real-world settings, however, agents cannot directly observe the underlying state and instead receive partial or noisy observations that do not satisfy the Markov property. Such problems are naturally modeled as partially observable Markov decision processes (POMDPs). By incorporating memory, an agent can summarize its interaction history into a latent state representation that captures information missing from the current observation, thereby approximating a Markovian representation and enabling standard RL methods to be applied more effectively in partially observable settings [29].

Recurrent neural networks (RNNs) [7, 13, 26], Transformers [2, 31, 48, 51, 53, 61], and state-space models (SSMs) [20, 21, 37, 38, 46, 49] have all been used as memory modules in model-free RL to summarize trajectory information and address POMDPs [47]. These memory models are usually trained jointly with the policy

Table 1: Parameter counts for two types of recurrent agents we used. All hidden layers have the same width. Reduction is relative to the fully trainable agent.

Model	Total	Trainable	Frozen	Reduction
PQN				
GRU	1,119,746	1,119,746	0	—
Frozen GRU	1,119,746	331,778	787,968	70.37%
LRU	858,114	858,114	0	—
Frozen LRU	858,114	331,778	526,336	61.34%
PPO				
GRU	1,385,221	1,385,221	0	—
Frozen GRU	1,385,221	597,253	787,968	56.88%
LRU	1,123,589	1,123,589	0	—
Frozen LRU	1,123,589	597,253	526,336	46.84%

or value function. However, this raises the question that is learning the memory module actually necessary? Several observations suggest that it may not always be. Tallec et al. [58] report that using an untrained Mixture-Density Recurrent Neural Network (MDRNN) from Ha and Schmidhuber [24] leads to only a 1.16% performance increase in a model-based RL setting. More recently, Gu et al. [22] show that freezing the recurrent and input matrices of S4 and S4D can preserve performance on certain tasks. These findings are closely related to Reservoir Computing (RC) [27, 39, 40], where a randomly initialized recurrent network serves as a fixed temporal feature extractor and only the output mapping is learned. Despite these promising observations, it remains unclear whether a frozen recurrent network can serve as effective memory for general POMDPs. We therefore ask: can an agent solve POMDPs using a randomly initialized recurrent network as memory while learning only the policy or value function, and what design choices make this possible? As shown in Tab. 1, freezing the recurrent memory can substantially reduce the number of trainable parameters, motivating a simpler and more parameter-efficient approach to partially observable RL.

To answer this question, we need to revisit Reservoir Computing (RC) mentioned before, because it is the most mature paradigm to design and leverage frozen recurrent networks. RC typically consists of two components, a reservoir and a readout. Generally, reservoir is a randomly initialized RNN and its weights remain frozen during training. The Echo State Network (ESN) [27] is a widely used architecture of RC. A key guarantee in ESN is the Echo State Property (ESP) which indicates that the hidden state should be independent of its initial state and only retain information about

*Corresponding author.

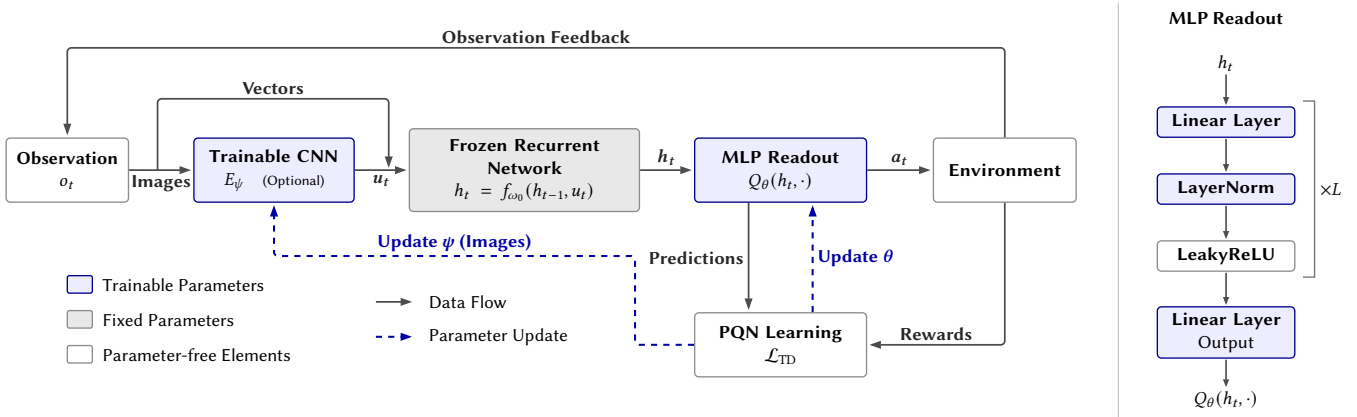


Figure 1: Overview of our PQN agent with frozen recurrence (left) and its nonlinear MLP Readout (right). We encode vector and visual observations differently. For visual control tasks, we train a Convolutional Neural Network (CNN) to extract and compress high-dimensional features. We optimize all trainable parameters using the PQN objective. The right panel shows the architecture of our simple MLP in detail. We also illustrate the architecture of the frozen recurrent network in Fig. 7.

the input history. However, recurrent networks used in RL are usually different from the architecture of ESN, and the ESP is not a necessary condition for recurrent networks design. Generalized reservoir computing (GRC) [33] offers a broader perspective of ESP, which verify that a suitable readout can restore input history even the hidden state retain dependence from initialization. This suggests that ESP is unnecessary for using frozen recurrent network as memory in recurrent agent.

Through extensive experiments, we find that, similar to [58], frozen recurrent networks can match the performance of trainable counterparts across a wide range of partially observable tasks, including long-horizon delayed-recall problems, discrete and continuous control, and pixel-based POMDPs. Performance degradation appears only in a small number of task-specific cases, where freezing the recurrent network leads to modest losses.

Here, we summarize the main contributions in this paper as follows: **(1)** We investigate whether frozen recurrent networks can serve as effective memory modules for solving POMDPs. **(2)** We propose a simple and effective agent architecture that uses a frozen recurrent network as memory, reducing the number of trainable parameters in RL by more than half. **(3)** Across a diverse set of POMDPs, we provide empirical evidence that learning the recurrent memory module is not always necessary for strong performance in partially observable RL. **(4)** We further show that agents with frozen recurrent memory can successfully learn visual encoders for high-dimensional partially observable control tasks.

2 Related Work

Recurrent Memory in Partially Observable RL. Recurrent agents learn history-dependent representations through LSTM-based architectures and online recurrent updates [12, 30]. GTxXL, CoBERL, HCAM, DTQN, AGaLiTe, and RATE use attention or recurrent memory to process past information [2, 6, 14, 34, 51, 53]. S5, FFM, and SHM provide structured recurrence and memory updates [36, 37, 45]. MERLIN and MRA combine recurrent networks with external memory to store past experiences and retrieve information

for decision making [15, 64]. These approaches still involve learning many parameters within their recurrent memory. We explore an alternative approach by adapting existing trainable recurrent architectures into frozen recurrent memories for solving POMDPs.

Reservoir Computing and RL. Reservoir computing separates a fixed recurrent backbone from a trainable readout [27, 40]. In POMDPs, ESN-based methods exploit this separation to construct history-dependent state representations for decision making [57, 65]. With trainable actor-critic readouts, ESNs can achieve competitive performance with learned recurrent memories on several memory-intensive tasks [41]. Related work has also explored fixed exponential memory traces for value prediction and control [11]. However, these studies primarily focus on reservoir-specific architectures rather than frozen versions of standard recurrent networks. We take a simpler approach: directly freeze the recurrent unit of conventional architectures and train only the downstream policy or value function. Despite this minimal modification, the resulting agents achieve strong performance across a diverse range of POMDPs.

RL with Frozen Recurrent Networks. Conventional frozen recurrent networks can also supply memory. Agents with randomly initialized frozen LSTMs can learn visual interval-timing tasks [9]. In robotic control, gated feature extraction combines an untrained GRU’s summary of preceding observations with the current observation for TD3 [3]. However, agents with frozen RNNs and GRUs failed on accumulating-towers, range-count, and exact-count tasks under the reported A2C setup [43]. However, these studies evaluate a limited range of POMDP environments. They also do not examine the theoretical basis of this phenomenon. We draw on GRC to guide the design of agents with frozen recurrent networks. Our simple architecture enables these agents to solve complex POMDPs.

3 Background

3.1 Reinforcement Learning (RL)

A Markov Decision Process (MDP) [56] is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, with state space $\mathcal{S}$, action space $\mathcal{A}$, state transition function $P(s_{t+1} | s_t, a_t)$, reward function $R(r_t | s_t, a_t)$ and discount factor γ . In RL, at each timestep t , the agent observes a state $s_t \in \mathcal{S}$ and selects an action a_t with respect to its policy $a_t \sim \pi(\cdot | s_t)$, which leads the agent to a new state $s_{t+1} \sim P(\cdot | s_t, a_t)$ while emitting a reward $r_t \sim R(\cdot | s_t, a_t)$. The Markov property states that the next state s_{t+1} depends only on the current state s_t and action a_t , as follows:

$$P(s_{t+1} | s_t, a_t) = P(s_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0).$$

The agent's objective is to find an optimal policy π^* that maximizes the expected discounted sum of returns

$$J(\pi, \mathcal{M}) = \mathbb{E}_{\pi, \mathcal{M}} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]. \quad (1)$$

3.2 Partially Observable RL and Recurrent Memory

A Partially Observable Markov Decision Process (POMDP) extends an MDP and is defined by the tuple $\mathcal{P} = \langle \mathcal{S}, \mathcal{A}, P, R, \Omega, O, \gamma \rangle$, where $\mathcal{S}, \mathcal{A}, P, R$, and γ follow the definitions above, Ω is the observation space, and $O(o_t | s_t)$ is the observation function. Unlike in an MDP, the agent cannot directly observe the latent state s_t and instead receives an observation $o_t \sim O(\cdot | s_t)$. In that case, a single observation o_t can not satisfy the Markov property, a trajectory $\mathbf{x}_t = (o_0, a_0, o_1, a_1, o_2, a_2, \dots, a_{t-1}, o_t)$ is used instead [56].

The objective for an agent with memory model f in a POMDP is therefore:

$$J(f, \pi, \mathcal{P}) = \mathbb{E}_{f, \pi, \mathcal{P}} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]. \quad (2)$$

A recurrent memory model implements f through a compact hidden state $\mathbf{h}_t$, updated from the previous state and current input [25, 47]:

$$\mathbf{h}_t = f_{\omega}(\mathbf{h}_{t-1}, o_t), \quad \mathbf{y}_t = R_{\theta}(\mathbf{h}_t). \quad (3)$$

Here, o_t is the current observation, and the readout R_{θ} maps the hidden state $\mathbf{h}_t$ to action values or policy parameters.

A Gated Recurrent Unit (GRU) uses input- and state-dependent gates to control memory updates and retention [7].

$$\begin{aligned} \tilde{\mathbf{h}}_t &= \tanh(\mathbf{W}o_t + \mathbf{U}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}), \\ \mathbf{h}_t &= (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t. \end{aligned} \quad (4)$$

Here, $\mathbf{W}$ and $\mathbf{U}$ are the input and recurrent weight matrices, $\mathbf{b}$ is a bias vector, and $\tilde{\mathbf{h}}_t$ is the candidate hidden state. The reset gate $\mathbf{r}_t$ controls past-state information in the candidate, while the update gate $\mathbf{z}_t$ controls how much of the candidate replaces the previous hidden state. Both gates apply a sigmoid to affine functions of o_t and $\mathbf{h}_{t-1}$; $\odot$ denotes elementwise multiplication.

A Linear Recurrent Unit (LRU) instead accumulates inputs in a complex-valued state through a diagonal linear recurrence [49],

$$\mathbf{h}_t = \mathbf{A}\mathbf{h}_{t-1} + \mathbf{B}o_t, \quad \mathbf{A} = \text{diag}(\lambda_1, \dots, \lambda_N). \quad (5)$$

The magnitudes of λ_i determine decay and their phases determine rotation, shaping memory retention. Real-valued output projections and nonlinear mixing may surround this linear cell.

3.3 Reservoir Computing and the Echo State Property

Reservoir Computing (RC) uses a dynamical system with randomly initialized recurrent networks to construct temporal features from an input sequence [27, 62]. Its hidden state evolves with the inputs, and a learned readout maps this state to predictions or control tasks. In a classical Echo State Network (ESN), the state update is

$$\begin{aligned} \mathbf{h}_t &= (1 - \alpha)\mathbf{h}_{t-1} \\ &+ \alpha \tanh(\mathbf{W}_{\text{in}}o_t + \mathbf{W}_{\text{rec}}\mathbf{h}_{t-1} + \mathbf{b}), \end{aligned} \quad (6)$$

where the leak rate $\alpha \in (0, 1]$ controls the mixture of the previous state and nonlinear update. The input weights $\mathbf{W}_{\text{in}}$, recurrent weights $\mathbf{W}_{\text{rec}}$, and bias $\mathbf{b}$ remain fixed [28]. A linear readout takes the form $\mathbf{y}_t = \mathbf{W}_{\text{out}}\mathbf{h}_t + \mathbf{b}_{\text{out}}$, although the readout can also be nonlinear.

The Echo State Property (ESP) describes the asymptotic independence of the hidden state from its initialization under a common input sequence [27, 68]. In the asymptotic forgetting formulation, trajectories $\mathbf{h}_t$ and $\mathbf{h}'_t$ driven by any admissible common input sequence converge from any pair of admissible initial states:

$$\lim_{t \rightarrow \infty} \|\mathbf{h}_t - \mathbf{h}'_t\| = 0. \quad (7)$$

For the leaky ESN, applying the 1-Lipschitz bound for $\tanh$ to two such trajectories yields

$$\|\mathbf{h}_t - \mathbf{h}'_t\|_2 \leq \kappa \|\mathbf{h}_{t-1} - \mathbf{h}'_{t-1}\|_2, \quad \kappa = (1 - \alpha) + \alpha \|\mathbf{W}_{\text{rec}}\|_2. \quad (8)$$

Thus $\kappa < 1$ is sufficient for contraction in the induced Euclidean operator norm. A recurrent spectral radius below one alone does not supply this norm bound for a nonlinear reservoir.

Forgetting an initial state is different from forgetting an input. Eq. 7 holds the driving sequence fixed; control requires distinguishing histories with different driving sequences. The distinction is explicit in a linear system with fixed $\mathbf{A}, \mathbf{B}$:

$$\mathbf{h}_t = \sum_{k=0}^t \mathbf{A}^{t-k} \mathbf{B}o_k. \quad (9)$$

3.4 Generalized Reservoir Computing

Generalized Reservoir Computing (GRC) shifts the ESP requirement from the reservoir state to the output [33]. It seeks a time-invariant (TI) transformation that extracts a response determined by input history, even when the hidden state retains initialization dependence. For a readout $\mathbf{y}_t = R_{\theta}(\mathbf{h}_t)$, independence of the output from initialization requires

$$\lim_{t \rightarrow \infty} \|R_{\theta}(\mathbf{h}_t) - R_{\theta}(\mathbf{h}'_t)\| = 0, \quad (10)$$

where $\mathbf{h}_t$ and $\mathbf{h}'_t$ receive identical inputs from different initial states. The resulting response has the form $\mathbf{y}_t^* = \Phi(o_t, o_{t-1}, \dots)$ after transients. TI means that the same mapping Φ applies at every time; the output can still vary with the input history.

A nonlinear readout can extract such a response by removing initialization-dependent components. For example, consider an oscillatory state $(X_t, Y_t) = (r_t \cos \varphi_t, r_t \sin \varphi_t)$ whose radius depends on input history while its phase retains initialization dependence.

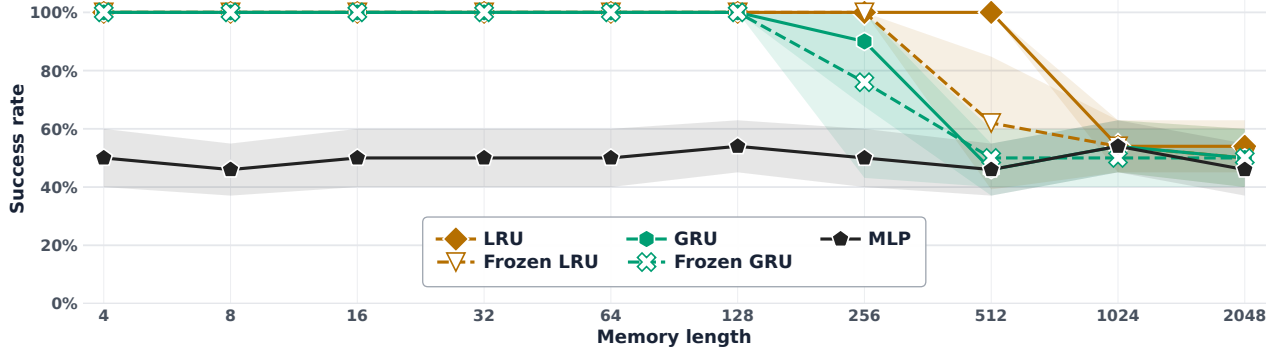


Figure 2: Evaluation results on MemoryChain. The shaded regions report the range of success rates across 5 seeds and 10 evaluation episodes per seed.

The transformation $X_t^2 + Y_t^2 = r_t^2$ removes the phase. GRC demonstrates related transformations using MLP readouts to recover past inputs from chaotic and physical reservoirs without ESP [33].

Frozen LRU as Linear Reservoir. Consider the LRU in Eq. 5, with $\mathbf{h}_t \in \mathbb{C}^N$ and fixed matrices $\mathbf{A}$ and $\mathbf{B}$. Its stable diagonal parameterization [49] takes the form

$$\lambda_j = \exp(-\exp(v_j) + i\varphi_j), \quad \rho = \max_j |\lambda_j| < 1, \quad (11)$$

where v_j and φ_j are finite real parameters and $i^2 = -1$. Each eigenvalue has magnitude $\exp(-\exp(v_j)) < 1$. For two trajectories initialized at arbitrary finite states $\mathbf{h}_0$ and $\mathbf{h}'_0$ and driven by the same inputs, subtracting their updates cancels $\mathbf{B}o_t$ at every step. Writing $\Delta \mathbf{h}_t = \mathbf{h}_t - \mathbf{h}'_t$ gives

$$\begin{aligned} \Delta \mathbf{h}_t &= \mathbf{A} \Delta \mathbf{h}_{t-1} = \mathbf{A}^t \Delta \mathbf{h}_0, \\ \|\Delta \mathbf{h}_t\|_2 &\leq \|\mathbf{A}^t\|_2 \|\Delta \mathbf{h}_0\|_2 = \rho^t \|\Delta \mathbf{h}_0\|_2. \end{aligned} \quad (12)$$

The equality $\|\mathbf{A}^t\|_2 = \rho^t$ follows from diagonality. Since $\rho < 1$, the initial-state difference vanishes as $t \rightarrow \infty$, establishing the ESP. For bounded inputs $\|o_t\|_2 \leq M$ extending into the past, the corresponding history response is

$$\mathbf{h}_t^* = \sum_{k=0}^{\infty} \mathbf{A}^k \mathbf{B} o_{t-k}, \quad \|\mathbf{h}_t^*\|_2 \leq \frac{M \|\mathbf{B}\|_2}{1 - \rho}. \quad (13)$$

Thus, the frozen LRU cell is a stable linear reservoir satisfying the ESP. It is a conventional reservoir.

Frozen GRU as General Reservoir. For a GRU, ESP depends on its parameters. Sufficient conditions for incremental stability ensure initial-state forgetting [4], but freezing the weights alone does not enforce these conditions. A two-dimensional construction illustrates how a frozen GRU can lack state ESP while admitting a readout with output ESP. Let $\mathbf{h}_t = (v_t, w_t)^\top \in [-1, 1]^2$ and $o_t \in [-1, 1]$. Set all gate weights and biases to zero, so $\mathbf{z}_t = \mathbf{r}_t = (1/2, 1/2)^\top$. In Eq. 4, choosing $\mathbf{W} = (1, 0)^\top$, $\mathbf{U} = \text{diag}(0, 4)$, and $\mathbf{b} = 0$ yields

$$\begin{aligned} v_t &= \frac{1}{2} v_{t-1} + \frac{1}{2} \tanh(o_t), \\ w_t &= \frac{1}{2} w_{t-1} + \frac{1}{2} \tanh(2w_{t-1}). \end{aligned} \quad (14)$$

The second coordinate has fixed points 0 and $w_* \in (0, 1)$, since $\tanh(2w) - w$ is positive just above zero and negative at one. Two

trajectories with identical v_0 and inputs but with $w_0 = 0$ and $w'_0 = w_*$ remain separated by w_* , violating state ESP. The first coordinate instead forgets its initialization. For updates starting at $t = 1$, unrolling gives $v_t = 2^{-t} v_0 + \sum_{k=0}^{t-1} 2^{-(k+1)} \tanh(o_{t-k})$. Extend the input sequence to $t \leq 0$ with any bounded prehistory and write $o_{\leq t} = (o_t, o_{t-1}, \dots)$. Then

$$\Phi(o_{\leq t}) = \sum_{k=0}^{\infty} 2^{-(k+1)} \tanh(o_{t-k}), \quad (15)$$

$$|v_t - \Phi(o_{\leq t})| \leq 2^{1-t} \rightarrow 0.$$

The bound follows from $|v_0| \leq 1$ and the geometric tail of the series. The linear readout $y_t = v_t$ therefore approaches a nonconstant function of input history. For a nonlinear readout $y_t = \ell(v_t)$, where $\ell(v) = \max(v, 0.01v)$, the 1-Lipschitz property of ℓ gives

$$|y_t - \ell(\Phi(o_{\leq t}))| \leq |v_t - \Phi(o_{\leq t})| \rightarrow 0.$$

Since ℓ is strictly increasing, the function of input history remains nonconstant. Both readouts discard the initialization-dependent coordinate, illustrating the GRC principle [33].

4 Experiments

Our experiments aim to answer the following research questions:

- **RQ1:** Can frozen recurrent networks retain input over long horizons in RL? (Sec. 4.2)
- **RQ2:** Can frozen recurrent networks solve vector observation discrete POMDPs? (Sec. 4.3)
- **RQ3:** Can frozen recurrent networks solve vector observation continuous POMDPs? (Sec. 4.4)
- **RQ4:** Can frozen recurrent networks solve high-dimensional image observed POMDPs? (Sec. 4.5)
- **RQ5:** Which learning matter in frozen recurrent networks? (sec. 4.6)

4.1 Experimental Setup

Baselines. Morad et al. [44] evaluated several memory architectures and found that GRU is the most performant. Recently, Wang et al. [63] and Lu et al. [38] found that LRU is the most powerful and efficient state-space memory architecture for RL. We therefore pick both of them as our baselines. For the frozen variants, we simply

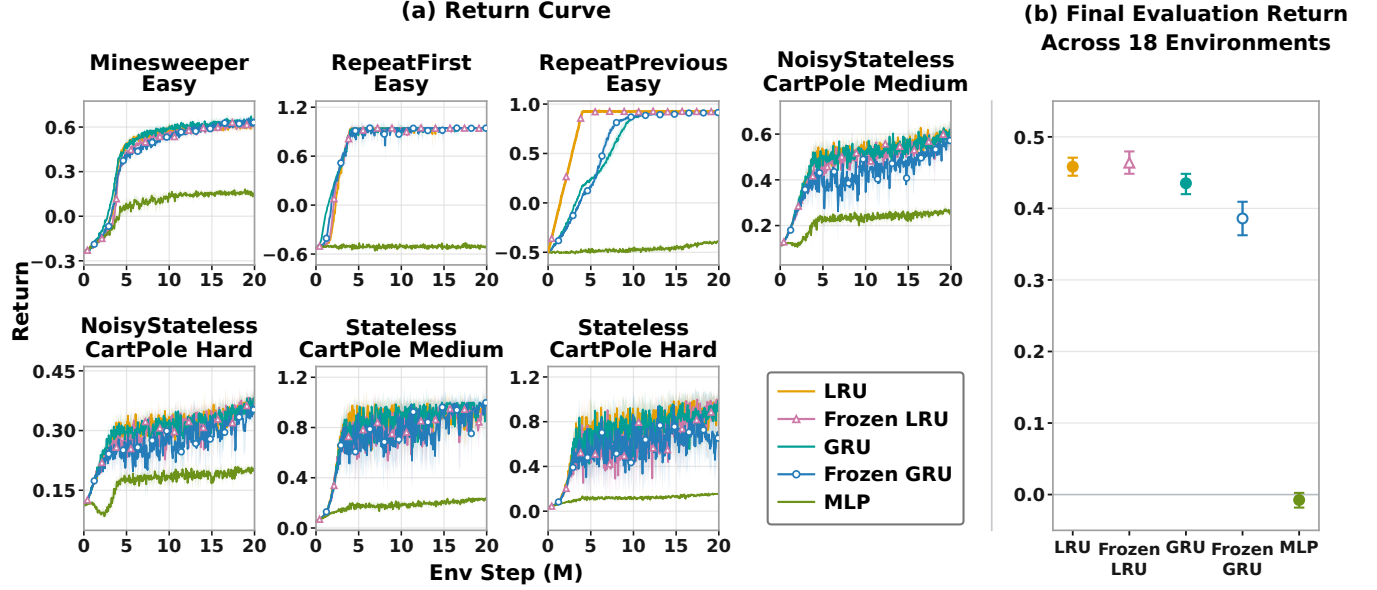


Figure 3: POPGym Results: (a) learning curves on seven selected environments, with 10 seeds per environment; (b) final evaluation returns over all 18 environments, with 10 seeds per environment and 10 evaluation episodes per seed.

Algorithm 1 Constructing and Using Frozen Recurrent Agents

Ensure: Agent and its trainable parameter set Θ_{learn}

$F_{\omega_0, \phi}$ is the recurrent memory network: ω_0 contains all recurrent unit parameters at initialization; ϕ denotes the trainable mixer parameters.

- 1: Initialize $F_{\omega_0, \phi}$ and R_θ
 - 2: Use FROZENSTEP for each recurrent unit
 - 3: $\Theta_{\text{learn}} \leftarrow \{\phi, \theta\}$
 - 4: $E \leftarrow \text{Identity}$
 - 5: **if** observations are images **then**
 - 6: Initialize CNN E_ψ ; set $E \leftarrow E_\psi$
 - 7: $\Theta_{\text{learn}} \leftarrow \Theta_{\text{learn}} \cup \{\psi\}$
 - 8: **end if**
 - 9: Use $E(o_t)$ as the input to $F_{\omega_0, \phi}$
 - 10: Keep ω_0 fixed; update only Θ_{learn} with the RL objective
 - 11: **return** agent $(E, F_{\omega_0, \phi}, R_\theta)$ and Θ_{learn}
 - 12: **function** FROZENSTEP($c, \omega_{c,0}, \mathbf{h}_{t-1}, \mathbf{u}_t, e_t$)
 - 13: c : a single recurrent unit (e.g., one GRU or LRU unit).
 - 14: $\omega_{c,0}$: unit c 's initialized parameter block in ω_0 .
 - 15: t : time step; $\mathbf{h}_{t-1}$: the cell's previous hidden state.
 - 16: $\mathbf{u}_t$: input features supplied to c at step t .
 - 17: $e_t \in \{0, 1\}$: episode start flag.
 - 18: **if** $e_t = 1$ **then**
 - 19: $\mathbf{h}_{t-1} \leftarrow \text{INITIALSTATE}(c)$ $\triangleright$ unit's reset state
 - 20: **end if**
 - 21: $\text{SG} \leftarrow \text{jax.lax.stop_gradient}$
 - 22: $\bar{\omega}_c \leftarrow \text{jax.tree.map}(\text{SG}, \omega_{c,0})$
 - 23: $(\mathbf{h}_t, \mathbf{x}_t^{\text{out}}) \leftarrow c_{\bar{\omega}_c}(\mathbf{h}_{t-1}, \mathbf{u}_t)$
 - 24: **return** $\mathbf{h}_t, \mathbf{x}_t^{\text{out}}$ $\triangleright$ Updated state, output features
 - 25: **end function**
-

apply Algo. 1 to the recurrent agent. Here, we use JAX-style pseudo code as an example.

Agent Architecture. We show our recurrent PQN agent architecture in Fig. 1. We use a simple MLP as readout, and each MLP consists of L blocks with a Linear Layer, a Layer Normalization [1], and LeakyReLU activation. We only stop the gradient of parameters in the recurrent unit, instead of cutting off gradient computation of recurrent models, so our agent still can learn a visual encoder to compress high-dimensional image observations. We update all the trainable parameters with the PQN learning target. Fig. 7 shows our detailed architecture of frozen recurrent networks. Each frozen recurrent unit is followed by a trainable feature mixer. For PPO, we only replace the Q-value readout MLP with same separate actor and critic MLP heads.

MemoryChain. MemoryChain is a classical one-bit delayed-recall task from bsuite [50]. We use the implementation from Gymnax [35] for faster rollout. We train eight delay configurations with Proximal Policy Optimization (PPO) [54], using 10M env steps and 5 seeds at each delay. Each seed receives ten evaluation episodes.

POPGym. Partially Observable Process Gym (POPGym) is a well-known benchmark for evaluating memory in partially observable RL [44]. We use POPJym [37], a JAX implementation of POPGym. Our evaluation includes six tasks: Minesweeper, NoisyStatelessCartPole, RepeatFirst, RepeatPrevious, and StatelessCartPole. Each task has three difficulty levels, giving 18 settings. We train each agent using PQN [18] with a budget of 20 million environment steps with 10 seeds per run.

POMDP MuJoCo Playground. We also hope to apply our framework to robot control. We therefore evaluate it on the DeepMind Control Suite tasks reimplemented in MuJoCo Playground [69]. We

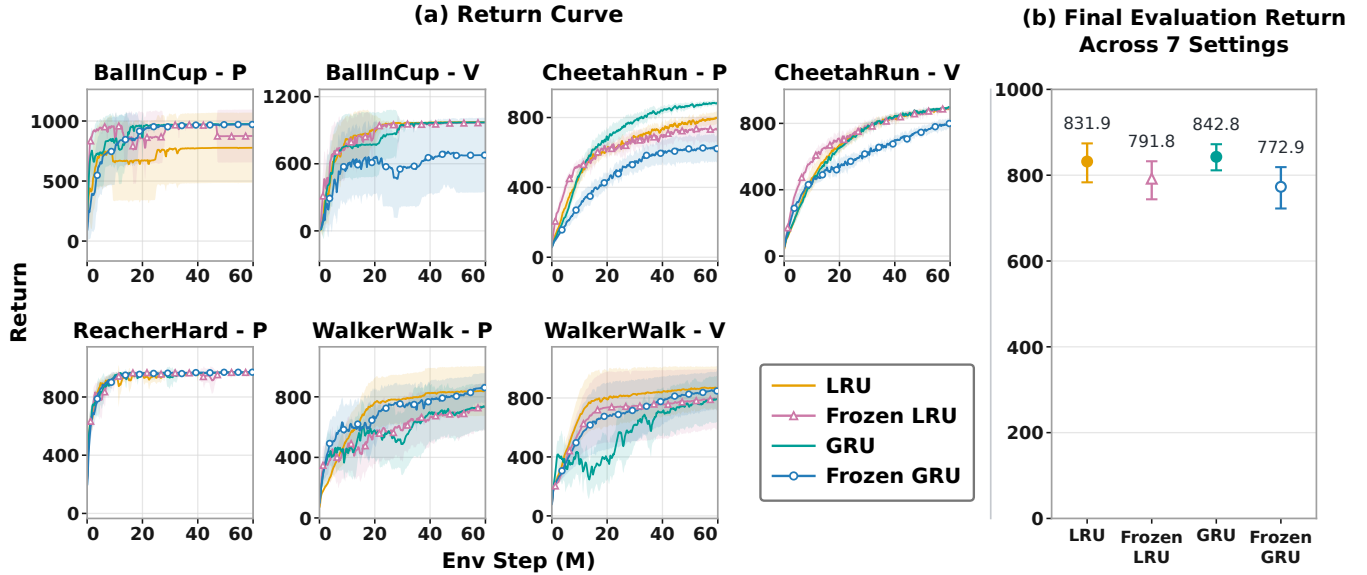


Figure 4: MuJoCo Playground Results: (a) learning curves across seven *position only* (P) and *velocity only* (V) settings, with means and confidence intervals over 10 seeds; (b) final evaluation return averaged across the seven settings over 10 seeds. Each seed is evaluated on 10 episodes after training.

use four environments: BallInCup, CheetahRun, ReacherHard, and WalkerWalk. We introduce partial observability by modifying the original observations. The *Positional Only* setting retains position information and hides velocity information. The *Velocity Only* setting retains velocity information and hides position information. Following the original paper, we use a budget of 60 million environment steps per run. We train with PPO [54] using a shared Actor-Critic trunk. Each configuration uses 10 seeds.

POPGym Arcade. We extend the evaluation to pixelated POMDPs with POPGym Arcade [63]. This benchmark provides both fully observable and partially observable Atari-style environments implemented in JAX. Our experiments cover the partially observable variants of BattleShip, CountRecall, MineSweeper, and Navigator. Agents receive $128 \times 128 \times 3$ RGB images as the observation from environments. To reduce computational cost and improve training efficiency, we train the models with PQN [18]. The training budget is 20 million environment steps per seed, with five training seeds per configuration.

Evaluation. We evaluate every agent after training. For each seed, we run 10 episodes and compute the mean return and variance to make the final evaluation return plots in Fig. 2, 3, 4 and 5.

4.2 RQ1: Long-Horizon Recall

We use MemoryChain to compare the recall performance of Frozen and trainable recurrent networks. In this environment, the agent receives a single-bit cue at the first time step of the episode, and then receives a sequence of zero observations for a number of delay steps. The agent receives a positive reward of +1 if it outputs the correct initial cue at the last time step, and -1 otherwise. We sweep the delay 2^k for $k \in \{2, \dots, 11\}$ and compare MLP, GRU,

LRU, frozen GRU and frozen LRU. Here, we introduce MLP as a memoryless baseline. We implement recurrent PPO with JAX [5] and Equinox [32] to accelerate the training process.

In Fig. 2, we plot the success rate of evaluation results. LRU achieves 100% success rate at delays up to 512 length, as expected for a state-of-the-art memory model in POMDPs. More surprisingly, frozen LRU gets 100% success rate at delays up to 256 length close to LRU. Similarly, the recall ability of frozen GRU also matches that of GRU, both achieve 100% success rate at delay of 128 length. Although the performance of frozen LRU and GRU does not exceed that of their trainable counterparts, these results still show the evidence that frozen recurrent networks have the ability to retain a cue over long horizons, indicating that simply freezing the parameters of the recurrent unit does not hurt too much of the memory retention of recurrent networks with RL objective.

4.3 RQ2: Vector Observation Discrete POMDPs

POPGym introduces partial observability to some simple RL tasks, making them more challenging and long-term memory demanding. For example, in MineSweeper, the agent can only observe the number of mines nearby instead of the actual grid, which requires the agent to remember which grids have been revealed before. We use 18 environments with approximately 1000 runs on POPGym to test the performance of Frozen GRU and LRU compared with their trainable counterparts as well as MLP. We plot seven representative selected learning curves in Fig. 3(a) and the final evaluation returns for all environments in Fig. 3(b). We present only informative results and exclude very simple tasks that almost all recurrent networks can solve.

Frozen recurrent networks achieve competitive performance while training substantially fewer parameters. As shown in Tab. 1,

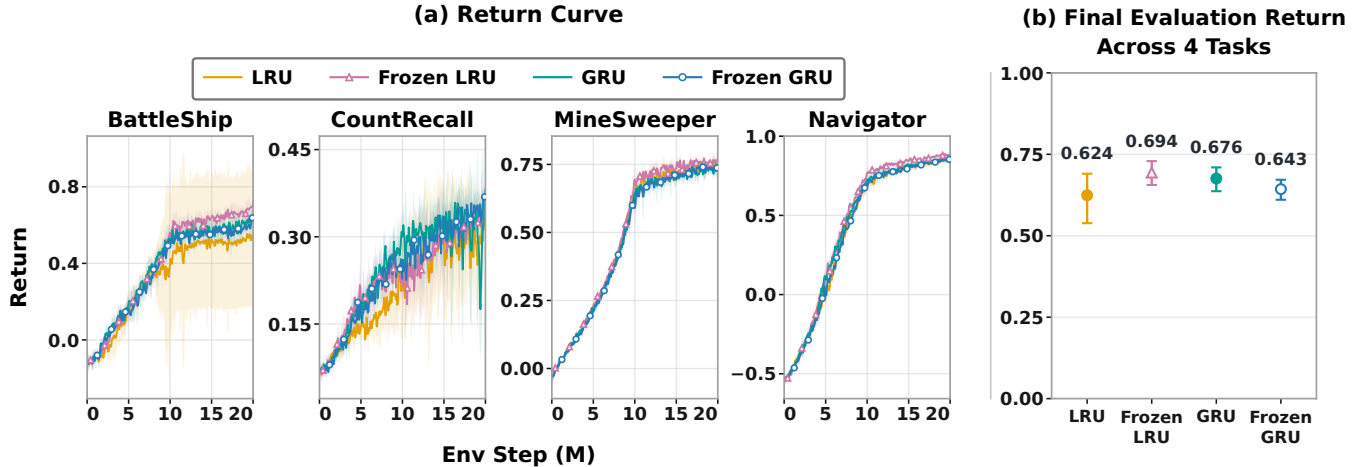


Figure 5: POPGym Arcade Results: (a) learning curves on four partially observable tasks with means and confidence intervals over 5 seeds; (b) final evaluation return averaged across the four tasks over 5 seeds. Each seed is evaluated on ten episodes.

Frozen LRU trains only approximately 40% of the total agent parameters, yet achieves a final evaluation return of 0.464, compared with 0.458 for LRU (Fig. 3(b)). This corresponds to a relative increase of 1.23%. Frozen GRU trains only approximately 30% of the total agent parameters and achieves a final evaluation return of 88.70% for GRU. These results show that updating all recurrent parameters is unnecessary for effective control on the evaluated POMDP tasks. In Fig. 3(a), the learning curves of Frozen LRU and LRU closely overlap on most tasks, indicating similar training progress and final performance. Although Frozen GRU often achieves lower returns than GRU during training, its final returns are comparable to GRU. These results demonstrate that frozen recurrent networks can serve as effective memory models to solve discrete POMDPs and achieve competitive performance.

4.4 RQ3: Vector Observation Continuous POMDPs

To further test the applicability of frozen recurrent networks in POMDPs, we also conducted experiments on partially observable continuous control tasks. We use MuJoCo Playground [69], the latest JAX reimplementation of MuJoCo [59]. It consists of several challenging tasks in which agents need to control a multi-joint dynamical body. We introduce *Position Only* and *Velocity Only* settings widely used in prior work for evaluating RL agents with memory [42, 47, 48], which are denoted as P and V in Fig. 4, respectively. We implement wrappers to mask the positional or velocity information in their observation.

Similarly, frozen recurrent networks also achieve competitive performance in partially observable continuous control tasks. In our PPO agents, Frozen LRU and Frozen GRU train approximately 53% and 43% of the total agent parameters, respectively (Tab. 1). Across the seven observation settings in Fig. 4(b), Frozen LRU achieves a final evaluation return of 791.8, compared with 831.9 for LRU, retaining 95% of LRU’s return. Frozen GRU achieves 772.9, compared with 842.8 for GRU, retaining 92% of GRU’s return. The learning curves in Fig. 4(a) show similar returns for all four agents on ReacherHard.

On WalkerWalk, Frozen GRU has higher mean training returns than GRU near the end of training under both *Position Only* and *Velocity Only* settings. The learning curves also show lower returns for Frozen LRU than LRU on WalkerWalk and for Frozen GRU than GRU on CheetahRun at the end of training. These results demonstrate that frozen recurrent networks can provide sufficient and efficient memory to solve such continuous POMDPs.

4.5 RQ4: Partially Observable Visual Control

In real-world scenarios, we usually regard most of the tasks as partially observable visual control tasks, in which agents need to make optimal decisions with image observations. With respect to Fig. 1 and Sec. 4.1, we can use gradients through a frozen recurrent network to learn a CNN as observation encoder to solve visual control POMDPs.

We evaluate our agents on POPGym Arcade [63], a JAX implemented extension of POPGym that provides various partially observable visual control tasks. It includes very challenging and high-dimensional visual POMDPs. For example, in the partially observable versions of BattleShip and MineSweeper, each observation reveals information only about the current grid cell. The agent must therefore extract features from these high-dimensional images, retain information about previously observed cells in memory, and retrieve it to guide decisions.

Notably, frozen recurrent networks also achieve competitive performance with learned visual representations. Across the four tasks in Fig. 5(b), Frozen LRU achieves a final evaluation return of 0.694, compared with 0.624 for LRU, representing a relative increase of 11.2% with lower variance. Frozen GRU achieves 0.643, retaining 95.2% of GRU’s final evaluation return of 0.676. In Fig. 5(a), the learning curves of all four agents approach similar returns on MineSweeper and Navigator. Frozen LRU reaches higher mean training returns than LRU on BattleShip, while the training curves on CountRecall fluctuate more but remain comparable at the end. These results demonstrate that frozen recurrent networks can work with learned CNN encoders to solve challenging and high-dimensional

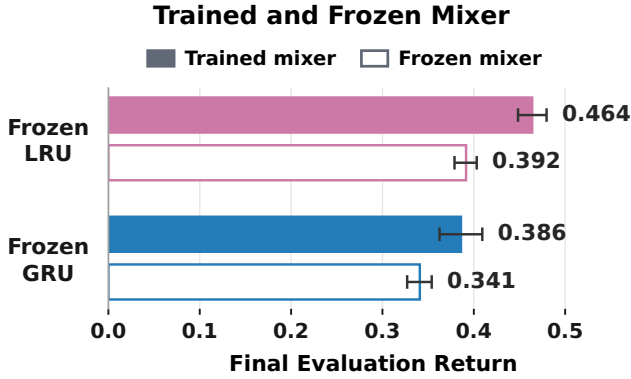


Figure 6: Comparison with trainable or frozen mixer. Bars show final evaluation return across 18 POPGym tasks, with 10 seeds and 10 evaluation episodes per seed.

visual control POMDPs. Moreover, Frozen LRU can even outperform its trainable counterpart, making it a promising baseline for partially observable RL.

4.6 RQ5: Learning the Mixers Matter

Modern recurrent architectures often use nonlinear feature mixer after each recurrent unit [21, 23, 49, 55] as well as Feed Forward Network (FFN) in transformer [8, 31, 61, 67]. In our experiments, we also adopt a similar design (Fig. 7) and conduct additional experiments to examine the effect of training these mixers while keeping the recurrent unit frozen.

We find that learning the mixers significantly improves performance for both Frozen LRU and Frozen GRU. As shown in Fig. 6, training the mixers increases the final evaluation return from 0.392 to 0.464 for Frozen LRU and from 0.341 to 0.386 for Frozen GRU, corresponding to relative improvements of 18.5% and 13.2%, respectively. Even the simple design used here (Fig. 7) substantially improves agent performance. These results suggest that learning the mixers contributes to the agent’s ability to adapt and use the memory provided by frozen recurrent networks. Modern large models use attention and complex nonlinear transformations to mix features [19, 60, 66]. Agents with frozen recurrent networks may also benefit from incorporating such designs into their mixers and readouts.

5 Discussion

Researchers often assume that recurrent RL requires training a memory model. Our experiments in Sec.4.2, 4.3, 4.4 and 4.5 challenge this assumption. We show that agents can solve the tested POMDPs with frozen recurrent networks and nonlinear MLPs and mixers can learn to use frozen recurrent networks as memory. We believe this approach offers an alternative solution for partially observable RL and may also benefit recurrent RL applications in robotic learning.

6 Limitations and Future Work

Our study has several limitations. First, we do not conduct a large-scale, exhaustive ablation study, and our evaluation covers a limited set of benchmarks. Computational constraints also restrict the

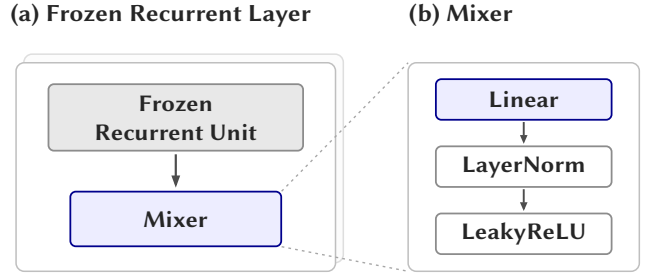


Figure 7: Frozen recurrent layer and mixer. (a) Each frozen recurrent unit is followed by a mixer. The overlapping out-lines represent recurrent layers stacked to form a recurrent network. (b) The mixer simply applies Linear, LayerNorm, and LeakyReLU. Blue denotes trainable parameters.

range of baselines considered. Future work should therefore include broader ablations, additional benchmarks, and stronger baselines to better assess the generality of our findings. Second, our GRC analysis relies on specific assumptions and establishes its result under a relaxed ESP condition. In practice, random initialization, optimization dynamics, and finite-sample effects introduce additional uncertainty, so trained agents may not always satisfy the theoretical conditions exactly. Future work could investigate how closely these conditions hold in practical RL settings and how deviations from them affect performance.

7 Conclusion

We investigated whether recurrent networks must be learned in partially observable RL. We first introduced a simple architecture that uses frozen recurrent networks as memory modules. Across diverse POMDPs, these frozen memories retain information over long horizons and achieve competitive performance with substantially fewer trainable parameters. We further showed that visual encoders can be trained effectively through frozen recurrent memories to solve high-dimensional visual control tasks. Overall, our results suggest that learning the recurrent module is not always necessary and frozen recurrent networks provide a simple, strong, and parameter-efficient baseline for recurrent RL.

AI Use Statement

We used OpenAI Codex with GPT-6 Astra for language polishing and LaTeX formatting. We also used it to write plotting code and code execution scripts. We did not retain complete prompt and conversation records. We take full responsibility for all the content, code, and results.

References

- [1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. [arXiv:1607.06450](https://arxiv.org/abs/1607.06450) <https://arxiv.org/abs/1607.06450>
- [2] Andrea Banino, Adria Puigdomenech Badia, Jacob C Walker, Tim Scholtes, Jovana Mitrovic, and Charles Blundell. 2022. CoBERL: Contrastive BERT for Reinforcement Learning. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=sRZ3GhmegS>
- [3] Peter Böhm, Pauline Pounds, and Archie C. Chapman. 2023. Feature Extraction for Effective and Efficient Deep Reinforcement Learning on Real Robotic Platforms. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 7126–7132. doi:10.1109/ICRA48891.2023.10160862

- [4] Fabio Bonassi, Marcello Farina, and Riccardo Scattolini. 2021. On the stability properties of Gated Recurrent Units neural networks. Revised manuscript, version 6. arXiv:2011.06806v6 doi:10.48550/arXiv.2011.06806
- [5] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Yash Katariya, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. JAX: composable transformations of Python+NumPy programs. <https://github.com/google/jax>
- [6] Egor Cherepanov, Aleksei Staroverov, Alexey K. Kovalev, and Aleksandr I. Panov. 2026. Recurrent Action Transformer with Memory. In *The Fourteenth International Conference on Learning Representations*. <https://openreview.net/forum?id=kByN4v0M3e>
- [7] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 1724–1734. doi:10.3115/v1/D14-1179
- [8] Soham De, Samuel L. Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. 2024. Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models. arXiv:2402.19427 <https://arxiv.org/abs/2402.19427>
- [9] Ben Deverett, Ryan Faulkner, Meire Fortunato, Gregory Wayne, and Joel Z. Leibo. 2019. Interval Timing in Deep Reinforcement Learning Agents. In *Advances in Neural Information Processing Systems*, Vol. 32. https://papers.nips.cc/paper_files/paper/2019/hash/2bf283c05b601f21364d052ca0ec798d-Abstract.html
- [10] Jonathan Dong, Ruben Ohana, Mushagh Rafayelyan, and Florent Krzakala. 2020. Reservoir computing meets recurrent kernels and structured transforms. In *Advances in Neural Information Processing Systems*, Vol. 33. 16785–16796. https://proceedings.neurips.cc/paper_files/paper/2020/hash/c348616cd8a86ee661c7c98800678fad-Abstract.html
- [11] Onno Eberhard, Michael Muehlebach, and Claire Vernade. 2025. Partially Observable Reinforcement Learning with Memory Traces. In *Proceedings of the 42nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 267)*. PMLR, 14934–14949. <https://proceedings.mlr.press/v267/eberhard25a.html>
- [12] Esraa Elelimy, Adam White, Michael Bowling, and Martha White. 2024. Real-Time Recurrent Learning using Trace Units in Reinforcement Learning. In *Advances in Neural Information Processing Systems*, Vol. 37. doi:10.52202/079017-0541
- [13] Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science* 14, 2 (1990), 179–211. doi:10.1207/s15516709cog1402_1
- [14] Kevin Esslinger, Robert Platt, and Christopher Amato. 2022. Deep Transformer Q-Networks for Partially Observable Reinforcement Learning. arXiv:2206.01078 [cs.LG] doi:10.48550/arXiv.2206.01078
- [15] Meire Fortunato, Melissa Tan, Ryan Faulkner, Steven Hansen, Adrià Puigdomènech Badia, Gavin Buttmore, Charles Deck, Joel Z. Leibo, and Charles Blundell. 2019. Generalization of Reinforcement Learners with Working and Episodic Memory. In *Advances in Neural Information Processing Systems*, Vol. 32. <https://proceedings.neurips.cc/paper/2019/hash/02ed812220b0705fabb868dbf17ea20-Abstract.html>
- [16] Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. 2017. Deep reservoir computing: A critical experimental analysis. *Neurocomputing* 268 (2017), 87–99. doi:10.1016/j.neucom.2016.12.089
- [17] Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. 2018. Design of deep echo state networks. *Neural Networks* 108 (2018), 33–47. doi:10.1016/j.neunet.2018.08.002
- [18] Matteo Gallici, Mattie Fellows, Benjamin Ellis, Bartomeu Pou, Ivan Masmitja, Jakob Nicolaus Foerster, and Mario Martin. 2025. Simplifying Deep Temporal Difference Learning. In *The Thirteenth International Conference on Learning Representations*. https://proceedings.iclr.cc/paper_files/paper/2025/hash/c23f3852601f6dd7f0b39223d031806f-Abstract-Conference.html
- [19] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. 2025. Gemma 3 Technical Report. arXiv:2503.19786 <https://arxiv.org/abs/2503.19786>
- [20] Albert Gu and Tri Dao. 2024. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. In *First Conference on Language Modeling*. <https://openreview.net/forum?id=tEYskwIVY2>
- [21] Albert Gu, Karan Goel, and Christopher Ré. 2022. Efficiently Modeling Long Sequences with Structured State Spaces. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=uYLFoz1v1AC>
- [22] Albert Gu, Ankit Gupta, Karan Goel, and Christopher Ré. 2022. On the Parameterization and Initialization of Diagonal State Space Models. In *Advances in Neural Information Processing Systems*, Vol. 35. 35971–35983. doi:10.52202/068431-2607
- [23] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. 2021. Combining recurrent, convolutional, and continuous-time models with linear state space layers. In *Advances in Neural Information Processing Systems*, Vol. 34. 572–585. <https://proceedings.neurips.cc/paper/2021/hash/05546b0e38ab9175cd905eebcb6bb76-Abstract.html>
- [24] David Ha and Jürgen Schmidhuber. 2018. Recurrent World Models Facilitate Policy Evolution. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. <https://proceedings.neurips.cc/paper/2018/hash/2de5d16682c3c35007e4e92982f1a2ba-Abstract.html>
- [25] Matthew Hausknecht and Peter Stone. 2015. Deep Recurrent Q-Learning for Partially Observable MDPs. In *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents*. <https://aaai.org/papers/11673-11673-deep-recurrent-q-learning-for-partially-observable-mdps/>
- [26] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780. doi:10.1162/neco.1997.9.8.1735
- [27] Herbert Jaeger. 2001. The “echo state” approach to analysing and training recurrent neural networks. GMD Report 148. German National Research Center for Information Technology. Corrected version with an erratum, January 26, 2010. <https://www.ai.rug.nl/minds/uploads/EchoStatesTechRep.pdf>
- [28] Herbert Jaeger and Harald Haas. 2004. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science* 304, 5667 (2004), 78–80. doi:10.1126/science.1091277
- [29] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101, 1–2 (1998), 99–134. doi:10.1016/S0004-3702(98)00023-X
- [30] Steven Kapturowski, Georg Ostrovski, John Quan, Rémi Munos, and Will Dabney. 2019. Recurrent Experience Replay in Distributed Reinforcement Learning. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=r1lyTjAqYX>
- [31] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 5156–5165. <https://proceedings.mlr.press/v119/katharopoulos20a.html>
- [32] Patrick Kidger and Cristian Garcia. 2021. Equinox: neural networks in JAX via callable PyTrees and filtered transformations. *Differentiable Programming workshop at Neural Information Processing Systems 2021* (2021). <https://arxiv.org/abs/2111.00254>
- [33] Tomoyuki Kubota, Yusuke Imai, Sumito Tsunegi, and Kohei Nakajima. 2024. Reservoir Computing Generalized. Version 1. arXiv:2412.12104v1 doi:10.48550/arXiv.2412.12104
- [34] Andrew K. Lampinen, Stephanie C. Y. Chan, Andrea Banino, and Felix Hill. 2021. Towards mental time travel: a hierarchical memory for reinforcement learning agents. In *Advances in Neural Information Processing Systems*, Vol. 34. <https://papers.nips.cc/paper/2021/hash/ed519dacc89b2bead3f453b0b05a4a8b-Abstract.html>
- [35] Robert Tjarko Lange. 2022. gymnasium: A JAX-based Reinforcement Learning Environment Library. <https://github.com/RobertTLange/gymnasium>
- [36] Hung Le, Dung Nguyen, Kien Do, Sunil Gupta, and Svetha Venkatesh. 2025. Stable Hadamard Memory: Revitalizing Memory-Augmented Agents for Reinforcement Learning. In *The Thirteenth International Conference on Learning Representations*. <https://openreview.net/forum?id=We5z3UEnUY>
- [37] Chris Lu, Yannick Schroecker, Albert Gu, Emilio Parisotto, Jakob Nicolaus Foerster, Satinder Singh, and Feryal Behbahani. 2023. Structured State Space Models for In-Context Reinforcement Learning. In *Advances in Neural Information Processing Systems*, Vol. 36. 47016–47031. doi:10.52202/075280-2036
- [38] Chenhao Lu, Ruizhe Shi, Yuyao Liu, Kaizhe Hu, Simon Shaojie Du, and Huazhe Xu. 2024. Rethinking Transformers in Solving POMDPs. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*. PMLR, 33089–33112. <https://proceedings.mlr.press/v235/lu24h.html>
- [39] Mantas Lukoševičius and Herbert Jaeger. 2009. Reservoir computing approaches to recurrent neural network training. *Computer Science Review* 3, 3 (2009), 127–149. doi:10.1016/j.cosrev.2009.03.005
- [40] Wolfgang Maass, Thomas Natschlager, and Henry Markram. 2002. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation* 14, 11 (2002), 2531–2560.
- [41] Kevin McKee. 2024. Reservoir Computing for Fast, Simplified Reinforcement Learning on Memory Tasks. arXiv:2412.13093 doi:10.48550/arXiv.2412.13093
- [42] Lingheng Meng, Rob Gorbet, and Dana Kulic. 2021. Memory-based Deep Reinforcement Learning for POMDPs. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 5619–5626. doi:10.1109/IROS51168.2021.9636140
- [43] James Mochizuki-Freeman, Sahaj Singh Maini, and Zoran Tiganj. 2023. Characterizing Neural Activity in Cognitively Inspired RL Agents during an Evidence Accumulation Task. In *2023 International Joint Conference on Neural Networks (IJCNN)*. IEEE. doi:10.1109/IJCNN54540.2023.10190998
- [44] Steven Morad, Ryan Kortvelesy, Matteo Bettini, Stephan Liwicki, and Amanda Prorok. 2023. POPGym: Benchmarking Partially Observable Reinforcement Learning. In *The Eleventh International Conference on Learning Representations*. <https://openreview.net/forum?id=chDrutUTs0K>

- [45] Steven Morad, Ryan Kortvelesy, Stephan Liwicki, and Amanda Prorok. 2023. Reinforcement Learning with Fast and Forgetful Memory. In *Advances in Neural Information Processing Systems*, Vol. 36. doi:10.52202/075280-3150
- [46] Steven Morad, Chris Lu, Ryan Kortvelesy, Stephan Liwicki, Jakob Foerster, and Amanda Prorok. 2024. Recurrent Reinforcement Learning with Memoroids. In *Advances in Neural Information Processing Systems*, Vol. 37. 14386–14416. doi:10.52202/079017-0459
- [47] Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. 2022. Recurrent Model-Free RL Can Be a Strong Baseline for Many POMDPs. In *Proceedings of the 39th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 162)*. PMLR, 16691–16723. <https://proceedings.mlr.press/v162/ni22a.html>
- [48] Tianwei Ni, Michel Ma, Benjamin Eysenbach, and Pierre-Luc Bacon. 2023. When Do Transformers Shine in RL? Decoupling Memory from Credit Assignment. In *Advances in Neural Information Processing Systems*, Vol. 36. 50429–50452. doi:10.52202/075280-2194
- [49] Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. 2023. Resurrecting recurrent neural networks for long sequences. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202)*. PMLR, 26670–26698. <https://proceedings.mlr.press/v202/orvieto23a.html>
- [50] Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvari, Satinder Singh, Benjamin Van Roy, Richard Sutton, David Silver, and Hado Van Hasselt. 2020. Behaviour Suite for Reinforcement Learning. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=rygf-kSYwH>
- [51] Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Sidhant Jayakumar, Max Jaderberg, Raphaël Lopez Kaufman, Aidan Clark, Seb Noury, Matthew Botvinick, Nicolas Heess, and Raia Hadsell. 2020. Stabilizing Transformers for Reinforcement Learning. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 7487–7498. <https://proceedings.mlr.press/v119/parisotto20a.html>
- [52] Matteo Pinna, Giacomo Lagomarsini, Andrea Ceni, and Claudio Gallicchio. 2026. ParLESN: Enabling parallel information processing in Reservoir Computing. In *Forty-third International Conference on Machine Learning*. <https://openreview.net/forum?id=NsjUKcONFR>
- [53] Subhojeet Pramanik, Esraa Elelimy, Marlos C. Machado, and Adam White. 2024. AGaLiTe: Approximate Gated Linear Transformers for Online Reinforcement Learning. *Transactions on Machine Learning Research* (2024). <https://openreview.net/forum?id=Ih6vOAHuvo>
- [54] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. arXiv:1707.06347 <https://arxiv.org/abs/1707.06347>
- [55] Jimmy T.H. Smith, Andrew Warrington, and Scott Linderman. 2023. Simplified State Space Layers for Sequence Modeling. In *The Eleventh International Conference on Learning Representations*. <https://openreview.net/forum?id=Ai8Hw3AXqks>
- [56] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press. <https://mitpress.mit.edu/9780262193986/reinforcement-learning/>
- [57] István Szita, Viktor Gyenes, and András Lőrincz. 2006. Reinforcement Learning with Echo State Networks. In *Artificial Neural Networks—ICANN 2006 (Lecture Notes in Computer Science, Vol. 4131)*. Springer, 830–839. doi:10.1007/11840817_86
- [58] Corentin Tallec, Léonard Blier, and Diviyan Kalainathan. 2018. Reproducing “World Models”: Is training the recurrent network really needed? Blog post. Accessed: 2026-10-08. <https://ctallec.github.io/world-models/>
- [59] Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. MuJoCo: A Physics Engine for Model-based Control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 5026–5033. doi:10.1109/IROS.2012.6386109
- [60] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 <https://arxiv.org/abs/2302.13971>
- [61] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, Vol. 30. https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html
- [62] David Verstraeten, Benjamin Schrauwen, Michiel D’Haene, and Dirk Stroobandt. 2007. An experimental unification of reservoir computing methods. *Neural Networks* 20, 3 (2007), 391–403. doi:10.1016/j.neunet.2007.04.003
- [63] Zekang Wang, Zhe He, Borong Zhang, Edan Toledo, and Steven Morad. 2026. Investigating Memory in Model-Free RL with POPGym Arcade. In *Proceedings of the 43rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 306)*. PMLR, 126692–126733. <https://proceedings.mlr.press/v306/wang26bx.html>
- [64] 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. 2018. Unsupervised Predictive Memory in a Goal-Directed Agent. arXiv:1803.10760 [cs.LG] doi:10.48550/arXiv.1803.10760
- [65] Kodai Yamashita and Tomoki Hamagami. 2022. Reinforcement Learning for POMDP Environments Using State Representation with Reservoir Computing. *Journal of Advanced Computational Intelligence and Intelligent Informatics* 26, 4 (2022), 562–569. doi:10.20965/jaciii.2022.p0562
- [66] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 Technical Report. arXiv:2505.09388 <https://arxiv.org/abs/2505.09388>
- [67] Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. 2024. Gated Linear Attention Transformers with Hardware-Efficient Training. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*. PMLR, 56501–56523. <https://proceedings.mlr.press/v235/yang24ab.html>
- [68] Izzet B Yildiz, Herbert Jaeger, and Stefan J Kiebel. 2012. Re-visiting the echo state property. *Neural Networks* 35 (2012), 1–9. doi:10.1016/j.neunet.2012.07.005
- [69] Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder Alexander Kahrs, Carmelo Sferrazza, Yuval Tassa, and Pieter Abbeel. 2025. Demonstrating MuJoCo Playground. In *Proceedings of Robotics: Science and Systems*. doi:10.15607/RSS.2025.XXI.020

Supplementary Material

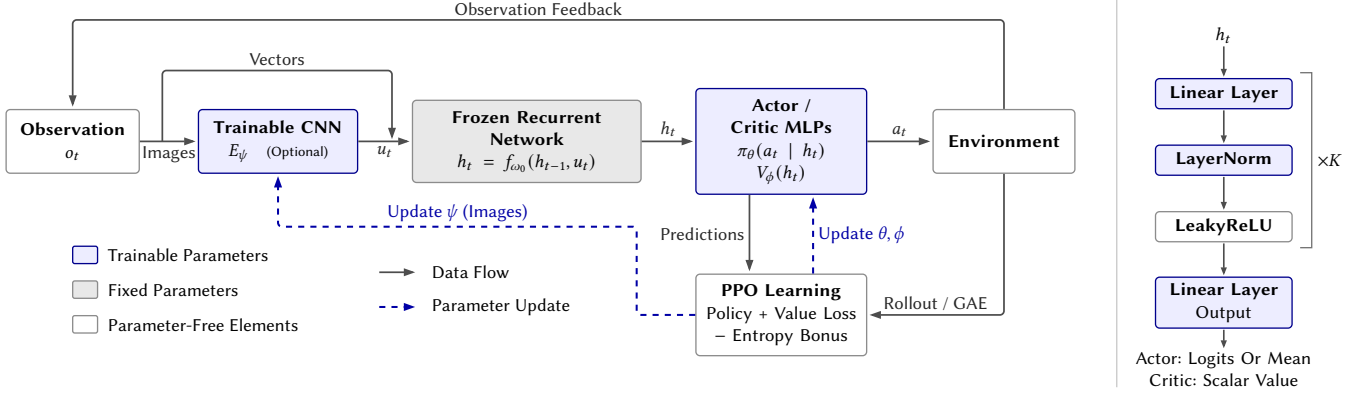


Figure A1: Overview of recurrent PPO agent with frozen recurrent networks. The frozen recurrent network supplies memory features to trainable actor and critic MLPs, which are optimized with PPO.

A PPO Agent

We show the recurrent PPO agent used in our MuJoCo Playground experiments in Fig. A1. It follows the PQN architecture in Fig. 1, with separate actor and critic MLPs in place of the PQN readout. The actor produces the policy, and the critic estimates the state value. We update the trainable parameters using PPO [54].

B Deep Reservoir Computing

Deep Reservoir Computing (Deep RC) [16] stacks reservoir layers to construct temporal features from input history and uses a trainable readout to learn from their hidden states. To further validate our conclusions, we conduct additional experiments with three Deep RC models: Deep Echo State Network (DeepESN), Structured Reservoir Computing (SRC), and Parallel Echo State Network (ParalESN).

B.1 Deep Echo State Network (DeepESN)

DeepESN extends a standard ESN by stacking multiple reservoir layers [17]. Let $\mathbf{h}_t^{(\ell)}$ denote the state of layer ℓ at time t . The first layer receives $\mathbf{u}_t^{(1)} = \mathbf{o}_t$, while each higher layer receives $\mathbf{u}_t^{(\ell)} = \mathbf{h}_t^{(\ell-1)}$. Each layer computes a candidate state and applies a leaky update:

$$\begin{aligned} \tilde{\mathbf{h}}_t^{(\ell)} &= \tanh(\mathbf{W}_{\text{in}}^{(\ell)} \mathbf{u}_t^{(\ell)} + \mathbf{W}_{\text{rec}}^{(\ell)} \mathbf{h}_{t-1}^{(\ell)} + \mathbf{b}^{(\ell)}), \\ \mathbf{h}_t^{(\ell)} &= (1 - \alpha^{(\ell)}) \mathbf{h}_{t-1}^{(\ell)} + \alpha^{(\ell)} \tilde{\mathbf{h}}_t^{(\ell)}. \end{aligned} \quad (\text{A1})$$

Here, $\mathbf{W}_{\text{in}}^{(\ell)}$ and $\mathbf{W}_{\text{rec}}^{(\ell)}$ are the fixed input and recurrent weight matrices, and $\mathbf{b}^{(\ell)}$ is a fixed bias. The leak rate $\alpha^{(\ell)} \in (0, 1]$ controls how much of the candidate replaces the previous state. For L_r reservoir layers, a typical readout uses

$$\mathbf{y}_t = R_\theta([\mathbf{h}_t^{(1)}; \dots; \mathbf{h}_t^{(L_r)}]), \quad (\text{A2})$$

where $[\cdot; \cdot]$ denotes concatenation and R_θ is the trainable readout. This gives the readout access to temporal features at different depths.

B.2 Structured Reservoir Computing (SRC)

SRC replaces dense random projections with structured random transforms [10]. For a layer with N state coordinates and input $\mathbf{u}_t \in \mathbb{R}^d$, the basic update is

$$\mathbf{h}_t = \frac{1}{\sqrt{N}} \phi(\mathbf{S}[\sigma_r \mathbf{h}_{t-1}; \sigma_{\text{in}} \mathbf{u}_t]). \quad (\text{A3})$$

Here, ϕ is an elementwise nonlinear activation, and σ_r and σ_{in} scale the previous state and current input. The structured projection $\mathbf{S} \in \mathbb{R}^{N \times (N+d)}$ is constructed from a square transform of the form

$$\mathbf{T}_m = \gamma \sqrt{m} \mathbf{H}_m \mathbf{D}_1 \mathbf{H}_m \mathbf{D}_2 \mathbf{H}_m \mathbf{D}_3, \quad (\text{A4})$$

where $\mathbf{H}_m$ is an orthogonally normalized Hadamard matrix, each $\mathbf{D}_j$ is diagonal with independent random signs in $\{-1, +1\}$, and γ sets the projection scale. The transform dimension $m \geq N + d$ is a power of two, and the corresponding $N \times (N + d)$ submatrix gives $\mathbf{S}$. The fast Hadamard transform evaluates this projection in $O(m \log m)$ operations without storing a dense matrix. The nonlinear state update still processes the sequence one time step at a time.

B.3 Parallel Echo State Network (ParalESN)

ParalESN separates linear recurrence from nonlinear mixing [52]. For one layer with input $\mathbf{u}_t$, the complex-valued state follows

$$\begin{aligned} \mathbf{h}_t &= \bar{\Lambda} \mathbf{h}_{t-1} + \alpha (\mathbf{W}_{\text{in}} \mathbf{u}_t + \mathbf{b}), \\ \bar{\Lambda} &= (1 - \alpha) \mathbf{I} + \alpha \Lambda. \end{aligned} \quad (\text{A5})$$

Here, Λ is a fixed complex diagonal matrix, $\mathbf{W}_{\text{in}}$ is the input matrix, $\mathbf{b}$ is a bias, $\mathbf{I}$ is the identity matrix, and $\alpha \in (0, 1]$ is the leak rate. The diagonal entries of the effective transition matrix $\bar{\Lambda}$ control decay and rotation of the state coordinates. The mixer then produces real-valued features:

$$\mathbf{q}_t = \tanh(\text{Re}(\mathbf{C} \mathbf{h}_t + \mathbf{c})), \quad (\text{A6})$$

where $\mathbf{C}$ represents a fixed convolution across state coordinates, $\mathbf{c}$ is its bias, and Re extracts the real part. These features feed the next block and the trainable readout. The mixer acts at each time step, outside the temporal recurrence in Eq. A5. The recurrence

therefore remains affine, so its updates compose associatively and can be evaluated by a parallel scan over a given input sequence.

C Reservoir Training on POPGym

We compare DeepESN, SRC, and ParalESN with their trainable variants on the 18 POPGym tasks. Fig. A2(a) shows the learning curves, and Fig. A2(b) summarizes final evaluation returns across all tasks.

D Detailed Evaluation Results

We report detailed final evaluation returns for POPGym, POPGym Arcade, and MuJoCo Playground. For each seed, we evaluate the trained agent over 10 episodes and compute its mean return. Each heatmap entry shows the mean and sample standard deviation across seeds. We use 10 seeds for POPGym and MuJoCo Playground, and 5 for POPGym Arcade.

D.1 POPGym

The POPGym heatmaps report evaluation results for 18 tasks and 11 model configurations. Figs. A3 and A4 report the final evaluation returns for the recurrent baselines and reservoir models, respectively. Both heatmaps use a shared color scale centered at zero. Frozen GRU and Frozen LRU retain trainable mixers and outer projections. We exclude Autoencode, Concentration, CountRecall, and MultiArmedBandit because of implementation bugs in the vector environments. ParalESN achieves a higher mean final evaluation

return than its trainable counterpart on six of the seven settings selected for the learning curves (Fig. A4).

D.2 POPGym Arcade and MuJoCo Playground

We report the final evaluation returns of GRU, LRU, and their frozen variants on four POPGym Arcade tasks and seven MuJoCo Playground settings in Fig. A5. We train the Arcade agents with PQN and the Playground agents with PPO. POPGym Arcade uses a separate implementation from vector POPGym, so we retain its CountRecall task.

E Model and Training Parameters

Tables A1 and A2 describe the recurrent agents and the CNN encoder used for POPGym Arcade. Here, d_o is the vector observation dimension, d_a is the number of discrete actions or the continuous action dimension, and d_{in} is the input feature dimension. We use $L = 1$ recurrent block for MemoryChain and $L = 2$ for the other benchmarks. Each readout has $K = 1$ hidden block for MemoryChain, $K = 2$ for POPGym and POPGym Arcade, and $K = 3$ for MuJoCo Playground. PPO uses separate actor and critic readouts with a shared recurrent trunk. All LeakyReLU activations use a negative slope of 0.01. The CNN uses no padding, and its output feature-map dimensions are listed as channels $\times$ height $\times$ width.

We use Adam with linear learning-rate decay. Tables A3–A6 list the main training hyperparameters. For MuJoCo Playground, we report the range of initial learning rates used across models and tasks.

Table A1: Architecture Of The Recurrent Agents.

Layer	Parameters	Activation
<i>Input And Shared Recurrent Trunk</i>		
Vector Input (PQN)	Observation And Previous One-Hot Action; $d_{in} = d_o + d_a$	None
Vector Input (PPO)	Observation Only; $d_{in} = d_o$	None
Image Input (PQN)	CNN Features And Previous One-Hot Action; $d_{in} = 512 + d_a$	None
Input Projection	Linear, $d_{in} \rightarrow 256$	None
<i>Recurrent Block</i>		
GRU	Input/Output Width $256 \rightarrow 256$	Sigmoid & Tanh
LRU	Input/Output Width $256 \rightarrow 256$	None
Mixer Linear Layer	Linear, $256 \rightarrow 256$	None
Mixer Normalization	LayerNorm Over 256 Features, Without Affine Parameters	LeakyReLU
<i>Trunk Output And Readout</i>		
Output Projection	Linear, $256 \rightarrow 256$	None
Readout Hidden Block	Linear, $256 \rightarrow 256$; LayerNorm With Affine Parameters; Repeat K Times	LeakyReLU
Q Output (PQN)	Linear, $256 \rightarrow d_a$; One Q-Value Per Action	None
Actor Output (PPO)	Linear, $256 \rightarrow d_a$; Action Logits Or Gaussian Means	None
Critic Output (PPO)	Linear, $256 \rightarrow 1$; Scalar State Value	None

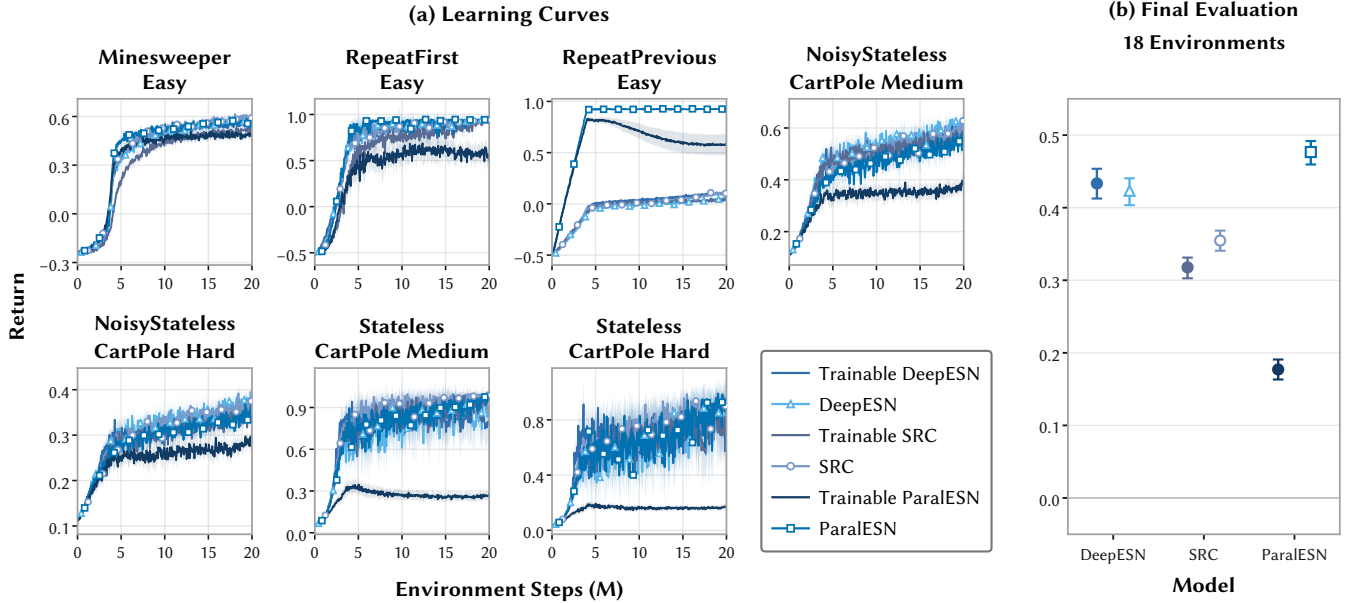


Figure A2: POPGym results for DeepESN, SRC, and ParaleSN and their trainable variants. (a) Learning curves on the selected seven tasks over ten seeds. (b) Final evaluation returns across all 18 POPGym tasks over ten seeds and ten evaluation episodes per seed

Table A2: CNN Observation Encoder For POPGym Arcade.

Layer	Parameters	Activation
Input Scaling	$128 \times 128 \times 3$ RGB Image	None
Conv2D 1	$3 \rightarrow 64$ Channels; 5×5 Kernel, Stride 2; Output $64 \times 62 \times 62$	LeakyReLU
MaxPool2D 1	2×2 Kernel, Stride 2; Output $64 \times 31 \times 31$	None
Conv2D 2	$64 \rightarrow 128$ Channels; 3×3 Kernel, Stride 2; Output $128 \times 15 \times 15$	LeakyReLU
MaxPool2D 2	2×2 Kernel, Stride 2; Output $128 \times 7 \times 7$	None
Conv2D 3	$128 \rightarrow 256$ Channels; 3×3 Kernel, Stride 2; Output $256 \times 3 \times 3$	LeakyReLU
MaxPool2D 3	3×3 Kernel, Stride 1; Output $256 \times 1 \times 1$	None
Conv2D 4	$256 \rightarrow 512$ Channels; 1×1 Kernel, Stride 1; Output $512 \times 1 \times 1$	LeakyReLU
Flatten	$512 \times 1 \times 1 \rightarrow 512$	None

		LRU	Frozen LRU	GRU	Frozen GRU	MLP	Mean Return
HigherLower	Easy	0.50±0.03	0.50±0.03	0.50±0.03	0.49±0.04	0.49±0.03	1.0
	Medium	0.50±0.02	0.50±0.02	0.50±0.01	0.49±0.02	0.50±0.02	
	Hard	0.50±0.02	0.49±0.02	0.50±0.02	0.50±0.02	0.50±0.02	
Minesweeper	Easy	0.70±0.13	0.83±0.13	0.76±0.18	0.76±0.16	0.25±0.10	0.5
	Medium	-0.24±0.10	-0.32±0.05	-0.35±0.07	-0.39±0.05	-0.25±0.07	
	Hard	-0.43±0.01	-0.44±0.02	-0.43±0.02	-0.43±0.03	-0.46±0.05	
NoisyStateless CartPole	Easy	0.99±0.01	0.99±0.02	0.99±0.01	0.98±0.02	0.45±0.08	0.0
	Medium	0.66±0.05	0.65±0.06	0.67±0.08	0.60±0.08	0.28±0.05	
	Hard	0.42±0.05	0.40±0.06	0.41±0.06	0.38±0.07	0.22±0.03	
RepeatFirst	Easy	1.02±0.00	1.02±0.00	1.02±0.00	1.02±0.00	-0.54±0.11	-0.5
	Medium	-0.50±0.19	-0.18±0.38	0.66±0.36	0.40±0.59	-0.50±0.13	
	Hard	-0.59±0.18	-0.65±0.21	-0.49±0.09	-0.55±0.20	-0.58±0.18	
RepeatPrevious	Easy	1.00±0.00	1.00±0.00	1.00±0.00	1.00±0.00	-0.36±0.04	-1.0
	Medium	1.00±0.00	0.98±0.04	-0.39±0.05	-0.50±0.03	-0.51±0.03	
	Hard	-0.23±0.19	-0.40±0.12	-0.48±0.03	-0.49±0.02	-0.50±0.02	
Stateless CartPole	Easy	1.00±0.00	1.00±0.00	1.00±0.00	1.00±0.00	0.46±0.07	
	Medium	0.97±0.09	0.98±0.05	0.97±0.09	1.00±0.00	0.24±0.03	
	Hard	0.99±0.03	0.99±0.02	1.00±0.01	0.70±0.31	0.17±0.03	

Figure A3: Detailed final evaluation returns for LRU, GRU, Frozen GRU, Frozen LRU and MLP on POPGym. Each entry shows the mean $\pm$ sample standard deviation over 10 seeds, with 10 evaluation episodes per seed

Table A3: PPO Hyperparameters For MemoryChain.

Hyperparameter	Value
Initial Learning Rate	3×10^{-4}
Final Learning Rate	10^{-10}
Discount γ	0.99
GAE λ	0.90
Gradient Clipping	0.5
PPO Clip	0.2
Entropy Coefficient	0
Value-Loss Coefficient	0.5
Number Of Environments	64
Rollout Length	1024
Minibatches Per Epoch	16
Update Epochs	4
Total Timesteps	10M

Table A4: PQN Hyperparameters For POPGym.

Hyperparameter	Value
Initial LR (Trainable RNNs)	10^{-5}
Initial LR (Fixed RNNs)	5×10^{-5}
Initial LR (MLP)	5×10^{-7}
Final Learning Rate	10^{-10}
Discount γ	0.99
TD(λ)	0.95
Gradient Clipping	10
Exploration ϵ	$1 \rightarrow 0.05$
Number Of Environments	16
Rollout Length	128
Minibatches Per Epoch	16
Update Epochs	4
Total Timesteps	20M

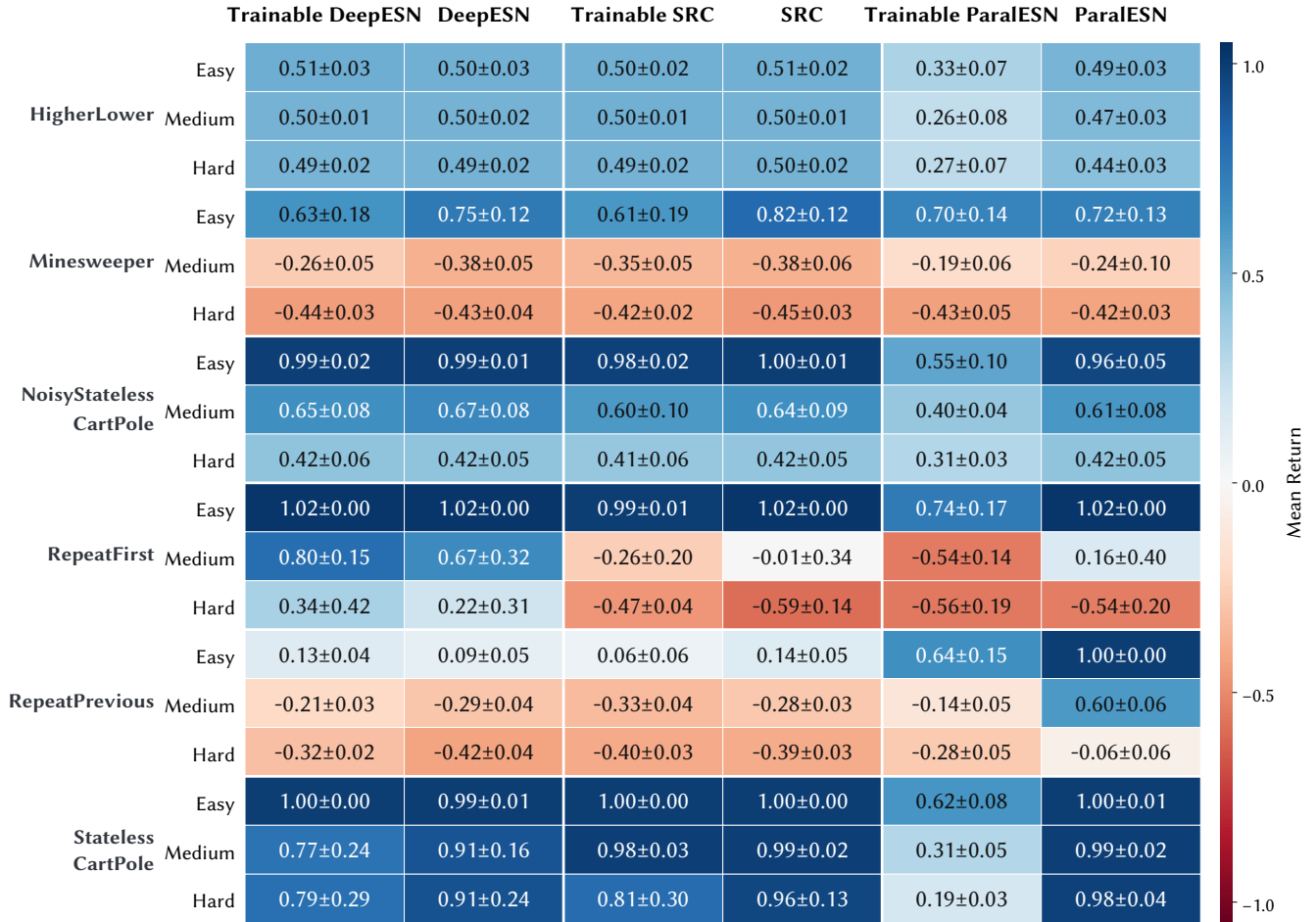
Figure A4: Detailed final evaluation returns for DeepESN, SRC, ParaESN, and their trainable variants on POPGym. Each entry shows the mean $\pm$ sample standard deviation over 10 seeds, with 10 evaluation episodes per seed.

Table A5: PQN Hyperparameters For POPGym Arcade.

Hyperparameter	Value
Initial LR (Trainable RNNs)	5×10^{-5}
Initial LR (Frozen RNNs)	3×10^{-4}
Final Learning Rate	5×10^{-7}
Discount γ	0.99
TD(λ)	0.65
Gradient Clipping	10
Exploration ϵ	$1 \rightarrow 0.05$
Number Of Environments	16
Rollout Length	128
Minibatches Per Epoch	4
Update Epochs	4
Total Timesteps	20M

Table A6: PPO Hyperparameters For MuJoCo Playground.

Hyperparameter	Value
Initial Learning Rate (Range)	$[1, 5] \times 10^{-5}$
Final Learning Rate	10^{-8}
Discount γ	0.995
GAE λ	0.95
Gradient Clipping	None
PPO Clip	0.3
Entropy Coefficient	0.01
Value-Loss Coefficient	0.5
Number Of Environments	16
Rollout Length	128
Minibatches Per Epoch	4
Update Epochs	4
Total Timesteps	60M

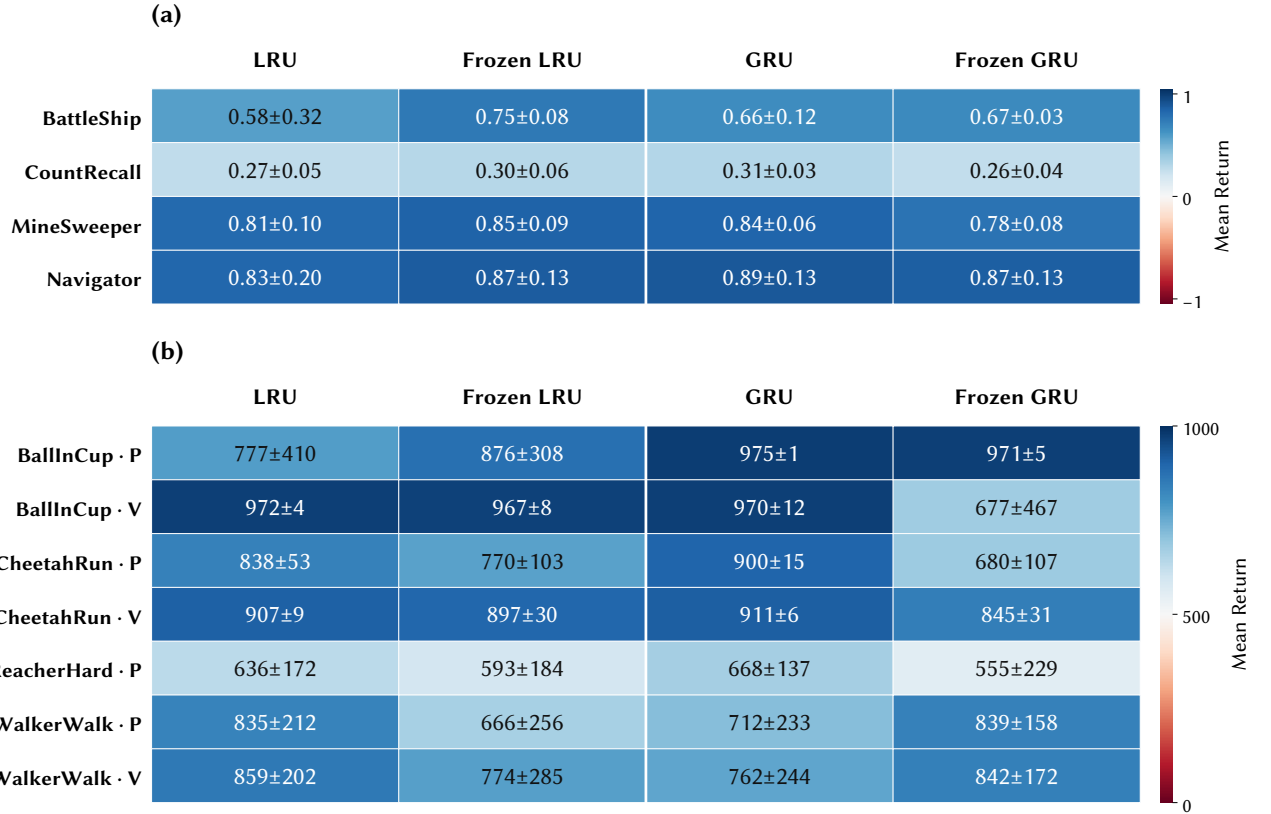


Figure A5: Detailed final evaluation returns for GRU, LRU, and their frozen variants on POPGym Arcade (top) and MuJoCo Playground (bottom). Each entry shows the mean $\pm$ sample standard deviation over 5 and 10 seeds, respectively, with 10 evaluation episodes per seed.