Exploring Non-Markov Environments Using Random Recurrent Memories

Ruo Yu Tao[1], **Kaicheng Guo**[1], **Cameron Allen**[2], **George Konidaris**[1]
ruo_yu_tao@brown.edu

[1]**Department of Computer Science, Brown University**
[2]**Center for Human-Compatible Artificial Intelligence, University of California, Berkeley**

Abstract

Exploring an environment requires a reinforcement learning agent to keep track of what it has already explored, which is difficult when observations do not fully reveal environment state. In the Markov setting, exploration algorithms have focused on achieving systematic coverage of observed states. These same methods have frequently been applied to the non-Markov setting, where they aim to achieve coverage over observations. However, decision-making in the non-Markov setting often depends on the agent's entire history, as opposed to only single observations. Unfortunately, achieving systematic coverage over the space of all trajectories is untenable: exploration over histories is exponentially expensive due to the dependence of the search space on the horizon. We therefore propose a new family of methods to featurize the agent's history with random recurrences. This produces finitely-sized random statistics, or *random recurrent memories*, over an agent's history, and we aim for coverage over these memories. We describe desirable properties for efficient history compression with random recurrences and propose a new architecture type, the tangent recurrent unit (TRU). We show that in a diverse suite of partially observable exploration tasks, tangent recurrent units, as well as other structured random recurrences, outperform popular methods that aim for observation coverage.

1 Introduction

Reinforcement learning agents need to consider *what* to explore over. Many things may be informative to sample for a return-maximizing agent, such as its transitions (Pathak et al., 2017; Tao et al., 2020; Henaff et al., 2022) or future tests (Littman & Sutton, 2001; Ramesh et al., 2022). The most popular form of exploration is to explore over the Markovian states of the environment (Bellemare et al., 2016; Burda et al., 2019; Lobel et al., 2023). In the more general setting under partial observability, we do not have access to the Markovian state and an agent must explore over other sampled statistics. Instead of states, an agent receives partial observations at every time step, and must condition on its history to inform decisions. Oftentimes Markovian exploration approaches have been applied to non-Markovian observations (Burda et al., 2019; Yin et al., 2021; Zamboni et al., 2024) with limited success. In this case, history information is discarded and not considered for exploration.

We investigate methods to explore efficiently over histories. Exploration over trajectories is difficult because it requires an agent to search over *sequences* rather than only single step statistics like state. This hardness is reflected when trying to take counts over trajectories; the space we are counting over grows exponentially with length of the trajectory. We propose to tackle trajectory exploration with structured, random fixed recurrences as a compression tool for exploration over histories. Since it can always be arbitrarily difficult to compress histories in POMDPs, we must rely on properly structured recurrences to efficiently disambiguate between histories. We propose parameterized recurrences with properties that will allow an agent to efficiently condense down histories to explore in the partially observable deep reinforcement learning setting. Based on these properties, we introduce the

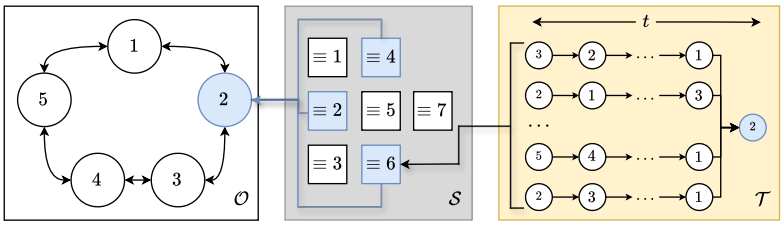


Figure 1: The Clock environment. **(Left)** Observation space exploration only gives coverage over the 5 observations. **(Center)** In state space, since observations are aliased (multiple states can produce observation 2), observation-based exploration leads to a coarse level of exploration over the modulo ground-truth states. **(Right)** Raw trajectories partition the exploration space too finely, potentially requiring an exponential (in time) number of samples for coverage.

tangent recurrent unit (TRU): a simple and principled linear recurrent function that allows agents to randomly (and efficiently) encode histories over a fixed parameterization. We use these random fixed recurrences as targets for density estimation to generate intrinsic rewards for deep reinforcement learning algorithms. Across a suite of difficult partially observable exploration tasks, structured random recurrences across histories generally outperform baseline algorithms that rely on novelty over observations, or model-based novelty estimates. These experiments demonstrate the need for algorithms to explore over histories.

2 Background

We consider the problem setting of Markov decision processes (MDPs) (Puterman, 1994) and its partially observable analog (Åström, Karl Johan, 1965; Kaelbling et al., 1998). An MDP is defined by its state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, state transition function $T : \mathcal{S} \times \mathcal{A} \to \Delta\mathcal{S}$, initial state distribution $p_0 \in \Delta\mathcal{S}$ and discount factor $\gamma \in [0, 1]$. States have the property that they are Markov: the state and action are a sufficient statistic to predict the next state and reward $P(s_{t+1}, r_t \mid s_t, a_t) = P(s_{t+1}, r_t \mid s_t, a_t, \ldots, s_0, a_0)$. An agent enacts its policy $\pi_{\mathcal{S}} : \mathcal{S} \to \Delta\mathcal{A}$ to maximize its discounted future return $g_t = \sum_{i=0}^{\infty} \gamma^i r_i$, where $r_i = R(s_i, a_i)$ with $s_i \in \mathcal{S}$, $a_i \in \mathcal{A}$. The expectation of which is the value function $V_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$. Following policy $\pi_{\mathcal{S}}$, an agent observes a distribution over states $\mu_{\pi_{\mathcal{S}}} \in \Delta\mathcal{S}$.

A partially observable Markov decision process (POMDP) extends MDPs by defining an observation function $\Phi : \mathcal{S} \to \Delta\mathcal{O}$. In general, observations $o \in \mathcal{O}$ are not Markov, and a statistic of the past observations and actions is required for accurate future predictions. We define sequences of observation-action pairs as the space of all possible trajectories $\mathcal{T} = (\mathcal{O} \times \mathcal{A})^* = \bigcup_{n=0}^{\infty} (\mathcal{O} \times \mathcal{A})^t$. As a shorthand, we denote an observation-action pair at time t as $x_t = (o_t, a_t)$ with $x_t \in \mathcal{X} = \mathcal{O} \times \mathcal{A}$. In POMDPs, an agent can either condition on its current observation to select actions $\pi_\Omega : \Omega \to \Delta\mathcal{A}$, or they can map an observed trajectory up to time t, $\tau_t \in \mathcal{T}$, to a next action $\pi : \mathcal{T} \to \Delta\mathcal{A}$ to maximize return. In order to efficiently condition over trajectories, we consider recurrences.

A recurrent function is a function over sequences with a recursive hidden state update. If the recurrent hidden state and input are in the sets $\mathcal{H}$ and $\mathcal{X}$ respectively, then a recurrent function F is defined as $F : \mathcal{H} \times \mathcal{X} \to \mathcal{H}$. Usually the recurrent hidden state space is over a fixed-size representation, like a vector over $\mathbb{R}^d$, to compress varying-length trajectories to a fixed size. Recurrent networks (Amari, 1972; Hochreiter & Schmidhuber, 1997; Cho et al., 2014; Jing et al., 2017) are a class of parameterized recurrent estimators that compress input sequences into hidden states that are fixed-length vectors: $f_\theta(\boldsymbol{x}_t, \boldsymbol{h}_t) = \boldsymbol{h}_{t+1}$, $\boldsymbol{h}_t, \boldsymbol{h}_{t+1} \in \mathbb{R}^d$. These estimators have been used extensively for state estimation (Bakker, 2001; Ni et al., 2022). In this work we consider how to use these function approximators for exploration in POMDPs.

3 Exploration under partial observability

An agent must decide what to explore over and how to do it. For example, coverage over the possible transitions an agent can experience is one option for exploration, and ensures that an agent experiences transitions that may lead to an improvement in its policy. This often incurs a heavy computational cost in modeling the environment (Pathak et al., 2017; Tao et al., 2020). Another option is count-based exploration (Bellemare et al., 2016; Ostrovski et al., 2017; Lobel et al., 2023): an agent is rewarded according to its epistemic uncertainty with regards to a state. This ensures coverage over the space of states.

Under partial observability, unless information on the belief over underlying Markovian states is available, exploration is either underspecified or requires samples that are exponential in the horizon. We make no assumptions about privileged access to information about the POMDP; we are only afforded the trajectory sequence we sample from the environment. In the Markov setting, since states are a sufficient statistic of an agent's past, exploration over the state space is enough to afford performance in exploration problems. However, under partial observability, an agent is also required to reason about its past for better decision making, and will likely need to explore over different trajectories to improve its policy.

Consider the Clock problem in Figure 1. In this fixed-horizon POMDP, observations are integers from 1 to 5, the two actions (+1 and -1) stochastically transition to either adjacent observations (with a loop between 5 and 1), and an agent gets a reward for observing a total sum that is divisible by 7. In order to maximize return, an agent must try out different trajectories which lead to different modulo 7 remainders. Let us consider count-based exploration in this environment by taking counts over state, observation, and trajectories.

Counts over the state space $\mathcal{S}$. For exploration in the Markov setting, an agent is incentivized to reach states it has not often seen. If $N(s)$ counts the number of times an agent has visited state $s \in \mathcal{S}$, then the agent sees an exploration bonus proportional to $N(s)^{-1/2}$. For count-based exploration over states in the Clock problem, an agent should try to visit sums with modulo 1 to 7, as shown in the center figure in Figure 1. In the non-Markov setting, without access to privileged state information (e.g. belief distributions), we have two alternatives: counts over observations and counts over trajectories.

Counts over the observation space $\mathcal{O}$. One can also consider incentivizing exploration by counting over visits to different observations. Comparing the left and center images in Figure 1, multiple states can map to the same observation. This means counting over observations is a fundamentally lossy way of counting over states—observing 2 does not guarantee the agent has visited all states that map to observation 2. In general, count-based exploration methods (even for POMDPs) aim for coverage of the state space. While using observation counts as an approximation of state counts can be beneficial in certain cases, we will show, through experimentation, that they often only work well in cases where observations themselves reveal much of the state.

Counts over the space of trajectories $\mathcal{T}$. While counts over trajectories is a more fine-grained measure than over observations, it can partition the input space too finely to be helpful for exploration. The right image in Figure 1 represents all the trajectories that could map to a single ground-truth state given it ends in observation 2.

If an agent needs to explore over all trajectories, the space that it needs to take counts over grows larger with the length of the trajectories. We can see this with visitation counts. The number of timesteps required to visit all trajectories of horizon T is bounded below by $|\mathcal{O}|^T|\mathcal{A}|^T$. With increasing trajectory lengths, this lower bound increases exponentially with the horizon T. Exploration over trajectories requires an exponentially large (in the horizon) number of samples without some form of compression over trajectories. In the next section we discuss different approaches to tackle this exponential blow up.

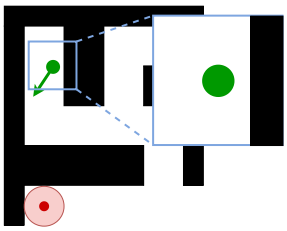


Figure 2: Visual PointMaze.

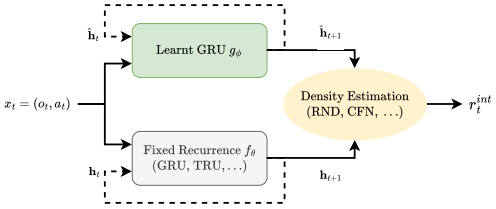


Figure 3: A general recurrent density estimation module for trajectory-based intrinsic reward r_t^{int}.

4 Compressing histories with random recurrences

Efficient exploration under partial observability requires a compression of trajectories to pare down the search space. Due to the nature and hardness of POMDPs, this reduction necessarily comes with trade-offs depending on the nature of the POMDP. One can construct a combinatorial lock POMDP that requires exploration over all trajectories, with no exponential blow up in exploring the state space (Liu et al., 2022). Nevertheless, there are reasonable priors we should assume about general POMDPs, and desirable properties of trajectory compressors for exploration that we now consider.

Recurrences for history compression. Instead, we can compress history so that approximate pseudocounts $\hat{N}$ (Bellemare et al., 2016) can generalize over the compression of history, and does not grow arbitrarily large with trajectory length. One form of history compression is through recurrence. In this work, we propose to use parameterized linear and non-linear recurrent networks $\boldsymbol{h}_{t+1} = f_\theta(\boldsymbol{h}_t, x_t)$ to compress trajectories into fixed-length histories. These fixed-length histories allow us to define pseudocounts over trajectories.

Non-stationary sequence compression in training. Most parameterized recurrent networks are learned dynamical systems like an RNN. The issue with learned systems is that their trajectory representation (e.g. the hidden state vector in an RNN) is non-stationary over training time. Trying to calculate pseudocounts over a moving representation can lead to ill-conditioned counts. To maintain stationarity, we propose to used fixed functions of histories, i.e. fixed recurrences.

Efficient disambiguation between trajectories. A memory compressor must group together histories to avoid paying the compounding cost over time in trajectory exploration with pseudocounts. Since pseudocounts require a distance metric to be defined over trajectories, these trajectory distances should ideally reflect things that are close in state space. One approach is to learn a memory compressor by optimizing auxiliary tasks (Ramesh et al., 2022; Allen et al., 2024). This again suffers from non-stationarity. Another approach is to use *structured* random recurrences for density estimation. While it may seem that a random projection of history would not be an efficient compression of history, in the next section we show that, with the right priors and structure, we can achieve more efficient compression with random recurrences.

4.1 Structured random recurrences for density-based exploration

The biggest concern with random fixed networks is the need for efficient history disambiguation for general POMDPs. In fact, for any given finite resolution history compressor that can disambiguate sequences up to length L, the combination lock POMDP (Liu et al., 2022) can be scaled such that $|\mathcal{A}|^T > L$ and the history compressor cannot disambiguate between trajectories. While there do exist simple compressions that are able to theoretically disambiguate all histories (Eberhard et al., 2025), these compressors are generally unable to compress histories with more complex dynamics.

We propose structured random recurrences with density estimation for intrinsic rewards to solve general partially observed environments. Density estimation exploration approaches approximate the

state visitation counts through prediction errors (Bellemare et al., 2016; Ostrovski et al., 2017; Burda et al., 2019; Lobel et al., 2023), and generates intrinsic reward. We visualize this proposed intrinsic reward module in Figure 3. This module has two parameterized models: a learnt recurrent function (in our case, a GRU) and a fixed, random recurrent function. We combine the outputs of these two functions for density estimation, which gives an intrinsic reward bonus. We consider what to use as a random fixed recurrence in this work. Since we cannot compress trajectories from all possible POMDPs with a finite resolution history compression, we have to rely on reasonable priors to reduce the space of trajectories to efficiently disambiguate between trajectories for exploration.

Consider the visual PointMaze environment (Fu et al., 2020; Radji, 2025) in Figure 2. The goal of an agent is to reach the goal (the red dot). Its state and action space is continuous, and its observations are cropped, top-down, agent-centric images of the agent's position. This requires localization based on the past history of observations to reach the goal position. We use this environment as an example for the different properties we might need for an efficient trajectory representation.

4.1.1 Fixed point under repeated inputs

Consider all the potential trajectories in visual PointMaze. A large portion of this space are trajectories where the agent is bumping into walls for extended periods of time. Repeated inputs generally include little information on the underlying state, as extended periods of the same input generally implies the state is not changing.

To avoid exploring these trajectories, we want our recurrent function to have a fixed point under the same input. For an input $\boldsymbol{x}$, our recurrence is $\boldsymbol{h}_{t+1} = f_\theta(\boldsymbol{h}, \boldsymbol{x})$, and denote f_θ^k as applying the function $f_\theta(\cdot, \boldsymbol{x})$ repeatedly k times. Then f_θ has a fixed point under input x when

$$\lim_{k\to\infty} f_\theta^k(\boldsymbol{h}, \boldsymbol{x}) = \boldsymbol{h}_x^\star, \text{ where } \boldsymbol{h}_x^\star = f_\theta(\boldsymbol{h}_x^\star, x). \tag{1}$$

This mechanism leaves out certain memory mechanisms that may be important. For example, counting for many time steps.

4.1.2 Bounded updates across time

Many density estimators use prediction error from random projections of the input (Burda et al., 2019; Lobel et al., 2023) as a way of estimating visitation densities. These predictions assume that the space of neural network outputs is smooth in the inputs: similar state inputs should admit random representations that are close. This allows the density estimates to generalize and predict similar inputs.

We propose a similar mechanism for trajectory-based exploration by predicting a statistic of the random fixed recurrency. To do so, we should consider an estimator that will predict this statistic. We use a standard GRU as our estimator for density estimation. If we consider the hidden state update of a GRU for each time step, that will bound how much the hidden state of a random fixed recurrent function can change so that our predictor can match the output. Due to the final $\tanh$ activations in a GRU, we have:

$$||\boldsymbol{h}' - \boldsymbol{h}||_2 \leq 2\sqrt{d},$$

where $\boldsymbol{h}'$ and $\boldsymbol{h}$ are two subsequent hidden states. This implies our random fixed recurrency should produce subsequent hidden states with distances that are less than $2\sqrt{d}$.

We would like updates to be well below this loose bound to prevent wildly swinging predictions across time. While a GRU as the random fixed recurrent network would satisfy this bound, in the next sections, we introduce a recurrent network architecture with tighter bounds on each update, which we show can improve exploration performance.

We consider what forms of recurrent units have bounded updates and have a fixed point under repeated inputs. We can show that gated recurrent units (GRUs) approximate these properties. For a more principled approach, we introduce a new recurrent unit that is a simple, principled recurrency and guarantees these properties.

4.2 Gated recurrent unit

GRUs are a natural candidate for random fixed recurrent networks for exploration. As a reminder, we want to use the prediction error between the output of a frozen GRU and a learned predictor network for density estimation. In this set up, if we use a model of the same architecture and size for both frozen and prediction networks, we are guaranteed our estimation model can at least *represent* the outputs of our random fixed recurrent model. This set up also trivially guarantees the fixed recurrence is bounded enough for prediction. However, we show in later sections that they are not necessarily the best for producing intrinsic rewards through density estimation.

Newly initialized GRUs are not guaranteed to have a fixed point under repeated inputs. Because of the orthogonal initialization of the recurrent kernel, we do not have a global contraction for fixed inputs as the spectral radius of the recurrent kernel is 1. The only guarantee we have is that a fixed point exists (Brouwer, 1911), but not necessarily will converge to it given any input.

4.3 Tangent recurrent unit

We introduce the tangent recurrent unit (TRU): a principled recurrent unit based on tangent-centered updates. TRUs are a parameterized recurrent network $\boldsymbol{h}_{t+1} = f_\theta(\boldsymbol{h}_t, \boldsymbol{x}_t)$ which updates its recurrent state vector (also called hidden state) with an input $\boldsymbol{x}_t$. This function happens in three steps:

$$\boldsymbol{z}_t = \tilde{\boldsymbol{B}}\boldsymbol{x}_t \tag{2}$$

$$\boldsymbol{z}_t^{\perp} = (\boldsymbol{I} - \boldsymbol{h}_{t-1}\boldsymbol{h}_{t-1}^{\intercal})\boldsymbol{z}_t \tag{3}$$

$$\boldsymbol{h}_t = \frac{\boldsymbol{h}_{t-1} + \boldsymbol{z}_t^{\perp}}{||\boldsymbol{h}_{t-1} + \boldsymbol{z}_t^{\perp}||_2} \tag{4}$$

where $\tilde{\boldsymbol{B}} \in \mathbb{R}^{d\times k}$ is a column-normalized. First, in Equation 2, we map the input $\boldsymbol{x}_t \in \mathbb{R}^k$ into $\mathbb{R}^d$ with a parameterized linear map $\tilde{\boldsymbol{B}}$. Next, in Equation 3, we project the mapped input $\boldsymbol{z}_t$ onto the orthogonal complement of the previous hidden state $\boldsymbol{h}_{t-1}$. Finally, in Equation 4, we add this projected vector $\boldsymbol{z}_t^{\perp}$ to the previous hidden state and normalize to produce the new hidden state. We visualize a step of this operation in Figure 5.

TRUs have many beneficial properties conducive for random fixed recurrent history compression. TRUs have a bounded update not dependent on hidden dimension ($||\boldsymbol{h}' - \boldsymbol{h}||_2 \leq 1$), due to the normalization step. With proper initialization, TRUs have a fixed point for almost all inputs, and update the hidden state in a bounded manner due to the normalization.

Theorem 1. *Let $\boldsymbol{x} \in \mathbb{R}^k$ be an input with $||\boldsymbol{x}||_2 = 1$, and define $\boldsymbol{z} := \tilde{\boldsymbol{B}}\boldsymbol{x} \in \mathbb{R}^d$. If $||\boldsymbol{z}||_2 > 0$, and if*

$$\boldsymbol{h}_+^* = \frac{\boldsymbol{z}}{||\boldsymbol{z}||_2} \quad \textit{and} \quad \boldsymbol{h}_-^* = -\boldsymbol{h}_+^*,$$

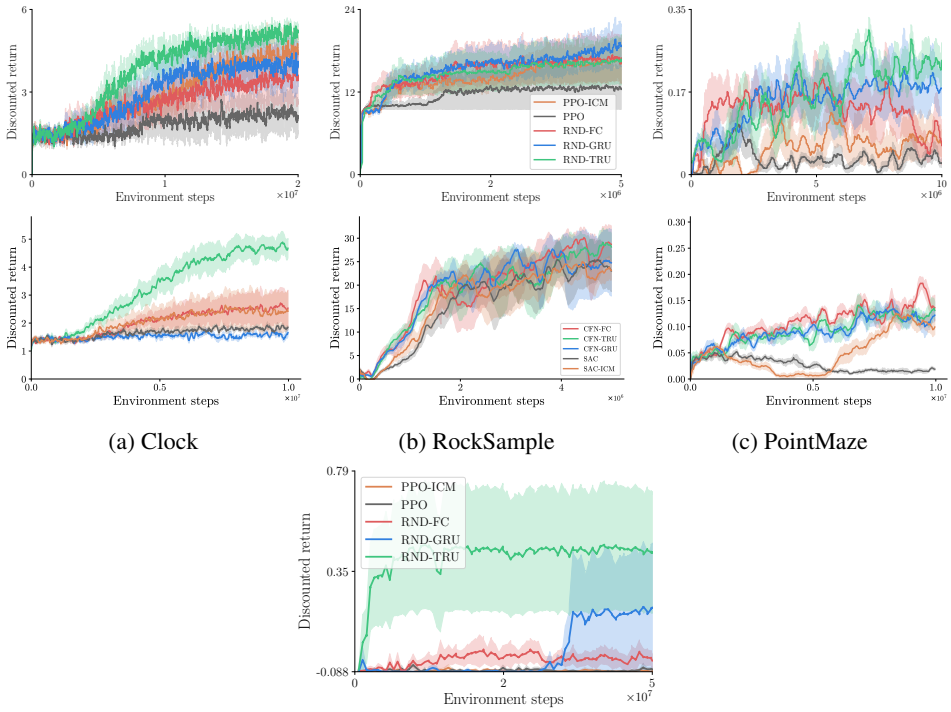
then repeated application of f_θ reaches the fixed point $\boldsymbol{h}_+^*$ for all $\boldsymbol{x}$ and all initializations of $\boldsymbol{h}_0 \neq \boldsymbol{h}_-^*$:

$$f_\theta(\boldsymbol{h}_+^*, \boldsymbol{x}) = \boldsymbol{h}_+^*.$$

Proof. See Section 7.1. □

A desirable property of recurrent functions is that the inputs are not commutative—the ordering of inputs should matter. Recurrent functions get around this with a transformation on the recurrent state; usually with a linear transformation that compounds over time. TRUs are not commutative, due to both the projection and normalization steps, since they change the hidden state with an input *relative* to the current hidden state.

Next, we conduct experiments to show that random fixed recurrences aid in exploration under partial observability.



(a) Clock (b) RockSample (c) PointMaze

(d) My Way Home

Figure 4: Performance across all four environments. Discounted episodic returns are reported for Clock, RockSample(11, 11), PointMaze, and My Way Home. The top row shows results with PPO as the base algorithm and the bottom row shows results with SAC as the base algorithm. Results are averaged across 5 seeds for all environments. Shaded regions show a 95% confidence interval.

5 Experiments

To test trajectory exploration capabilities, we introduce a suite of reinforcement learning environments that focus on memory-based exploration. This benchmark is entirely written in JAX (Bradbury et al., 2018), allowing for fast and scalable experimentation. We test different exploration algorithms on four environments that capture diverse forms of partial observability and differing levels of exploration difficulty.

Clock The Clock environment was introduced in Section 3. The agent receives reward at the end of an episode only when the sum of its observations is divisible by a chosen integer. Because reward depends on the full sequence rather than on any individual observation, an agent must explore over entire trajectories to discover the modular structure that yields rewards.

RockSample RockSample[n, k] (Smith & Simmons, 2004) places an agent in an $n \times n$ grid containing k rocks of unknown quality (good or bad). The agent observes its own position and receives noisy sensor readings about rock quality. It must navigate and sample the good rocks before exiting on the right edge of the grid.

PointMaze This is a continuous control navigation task (Fu et al., 2020; Radji, 2025) where an agent must traverse a maze to reach a random goal location. The observation is a 10×10 agent centric image as shown in Figure 2. Since the agent cannot localize itself from any single frame, it must do so through its history.

My Way Home A challenging visual first-person maze exploration problem introduced in ViZ-Doom (Wydmuch et al., 2019), and hardware accelerated with JAX (Bradbury et al., 2018) in

JAXenstein (Tao & Konidaris, 2026). An agent randomly spawns in a location and has to navigate a maze to reach a fixed goal position.

5.1 Experiment setup

We test exploration with random fixed recurrent memories this on two exploration algorithms: proximal policy optimization (PPO) (Schulman et al., 2017) with random network distillation (RND) (Burda et al., 2019) and soft actor-critic (SAC) (Haarnoja et al., 2018) with coin flipping networks (CFN) (Lobel et al., 2023).

In standard instantiations of these algorithms, the exploration target is a fixed function of the current observation. In our recurrent variants, we replace this target with two variations of random fixed recurrences over the observation-action history: a randomly initialized GRU (RND-GRU, SAC-GRU) and the tangent recurrent unit (RND-TRU, SAC-TRU) described in Sections 4.2 and 4.3 respectively. The predictor network is a learned GRU trained to predict the output associated with these fixed recurrent targets and we use its prediction error as an intrinsic reward.

We compare these against three baselines. First, we run the base algorithm without any exploration bonus (PPO, SAC). Second, we run the exploration algorithms where the intrinsic reward is computed with only the current observation-action pair (RND-FC, CFN-FC). Third, we compare against a recurrent intrinsic curiosity module (ICM) (Pathak et al., 2017) added to PPO and SAC (PPO-ICM, SAC-ICM). We sweep intrinsic reward scales and algorithm-specific loss coefficients for each method, and report performance over 5 random seeds in Figure 4. Details of all experiments are given in Section 8.

5.2 Results

Figure 4 shows the performance of all methods on our exploration benchmark. Overall, algorithms that explore histories consistently outperform the ones that only explore over partially observed observations. The only environment where the observation-based exploration matches performance with the memory-based counterparts is in RockSample(11, 11). This is likely due to the structure of RockSample observations: by themselves they reveal much about the state of the system. Clock, PointMaze and My Way Home are three environments which all require trajectory exploration as their observations do not reveal enough about state. TRUs perform particularly well in both the Clock and My Way Home environments, and has matching performance on PointMaze, suggesting that it may be a better candidate for trajectory-focused exploration.

6 Conclusion

Exploration over an agent's history is necessary in most interesting cases of partial observability. We present a family of methods to featurize the agent's history with recurrences, producing fixed-sized random recurrent memories for an agent to explore over. To incorporate this into a reinforcement learning algorithm, we propose to use these random recurrent memories for density estimation to output intrinsic rewards. For this history compression to be viable with density estimation, we test recurrent structures that are both bounded and have a fixed point over repeated inputs. With these properties in mind, we introduce a more principled recurrent function, the tangent recurrent unit, that gives guarantees on these properties. Through deep reinforcement learning experiments, we show that conditioning on trajectories in these well-known exploration algorithms leads to more effective exploration in hard partially observable exploration tasks.

References

Cameron Allen, Aaron Kirtland, Ruo Yu Tao, Sam Lobel, Daniel Scott, Nicholas Petrocelli, Omer Gottesman, Ronald Parr, Michael L. Littman, and George Konidaris. Mitigating partial observability in sequential decision processes via the lambda discrepancy. In Advances in Neural Information Processing Systems, volume 37, 2024.

Shun-ichi Amari. Learning patterns and pattern sequences by self-organizing nets of threshold elements. IEEE Transactions on Computers, C-21(11):1197–1206, 1972.

Bram Bakker. Reinforcement learning with long short-term memory. In T. Dietterich, S. Becker, and Z. Ghahramani (eds.), Advances in Neural Information Processing Systems, volume 14. MIT Press, 2001.

Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Rémi Munos. Unifying count-based exploration and intrinsic motivation. In Proceedings of the 30th International Conference on Neural Information Processing Systems, NIPS'16, pp. 1479–1487, 2016.

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. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

Luitzen EJ Brouwer. Über abbildung von mannigfaltigkeiten. Mathematische Annalen, 71(1):97–115, 1911.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In International Conference on Learning Representations, 2019.

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

Onno Eberhard, Michael Muehlebach, and Claire Vernade. Partially observable reinforcement learning with memory traces. In Forty-second International Conference on Machine Learning, 2025.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In International Conference on Machine Learning, 2018.

Mikael Henaff, Roberta Raileanu, Minqi Jiang, and Tim Rocktäschel. Exploration via elliptical episodic bonuses. In Advances in Neural Information Processing Systems. Curran Associates, Inc., 2022.

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

Li Jing, Yichen Shen, Tena Dubcek, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. Tunable efficient unitary neural networks (eunn) and their application to rnns. In Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17, pp. 1733–1741. JMLR, 2017.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. Artificial Intelligence, 101(1):99–134, 1998. ISSN 0004-3702.

Michael Littman and Richard S Sutton. Predictive representations of state. In T. Dietterich, S. Becker, and Z. Ghahramani (eds.), Advances in Neural Information Processing Systems, volume 14. MIT Press, 2001.

Qinghua Liu, Alan Chung, Csaba Szepesvari, and Chi Jin. When is partially observable reinforcement learning not scary? In Po-Ling Loh and Maxim Raginsky (eds.), Proceedings of Thirty Fifth Conference on Learning Theory, volume 178 of Proceedings of Machine Learning Research, pp. 5175–5220. PMLR, 02–05 Jul 2022.

Sam Lobel, Akhil Bagaria, and George Konidaris. Flipping coins to estimate pseudocounts for exploration in reinforcement learning. In Proceedings of the 40th International Conference on Machine Learning, ICML'23, 2023.

Chris Lu, Jakub Kuba, Alistair Letcher, Luke Metz, Christian Schroeder de Witt, and Jakob Foerster. Discovered policy optimisation. Advances in Neural Information Processing Systems, 35:16455–16468, 2022.

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

Georg Ostrovski, Marc G. Bellemare, Aäron van den Oord, and Rémi Munos. Count-based exploration with neural density models. In Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17, pp. 2721–2730. JMLR, 2017.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In International Conference on Machine Learning (ICML), 2017.

Martin L. Puterman. Markov Decision Processes. Wiley, 1994.

Waris Radji. Pointax: Jax-native pointmaze environment, 2025. URL https://github.com/riiswa/pointax.

Aditya Ramesh, Louis Kirsch, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Exploring through random curiosity with general value functions. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), Advances in Neural Information Processing Systems, volume 35, pp. 18733–18748. Curran Associates, Inc., 2022.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

Trey Smith and Reid Simmons. Heuristic search value iteration for pomdps. In Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence, UAI'04, 2004.

Ruo Yu Tao and George Konidaris. Jaxenstein: Accelerated benchmarking for first-person environments, 2026.

Ruo Yu Tao, Vincent Francois-Lavet, and Joelle Pineau. Novelty search in representational space for sample efficient exploration. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), Advances in Neural Information Processing Systems, volume 33, pp. 8114–8126. Curran Associates, Inc., 2020.

Marek Wydmuch, Michał Kempka, and Wojciech Jaśkowski. ViZDoom Competitions: Playing Doom from Pixels. IEEE Transactions on Games, 11(3):248–259, 2019. The 2022 IEEE Transactions on Games Outstanding Paper Award.

Haiyan Yin, Jianda Chen, Sinno Jialin Pan, and Sebastian Tschiatschek. Sequential generative exploration model for partially observable reinforcement learning. Proceedings of the AAAI Conference on Artificial Intelligence, 35(12):10700–10708, May 2021.

Riccardo Zamboni, Duilio Cirino, Marcello Restelli, and Mirco Mutti. The limits of pure exploration in pomdps: When the observation entropy is enough. Reinforcement Learning Journal, 2:676–692, 2024.

Åström, Karl Johan. Optimal Control of Markov Processes with Incomplete State Information I. 10: 174–205, 1965. ISSN 0022-247X.

Supplementary Materials

The following content was not necessarily subject to peer review.

7 Tangent recurrent unit details

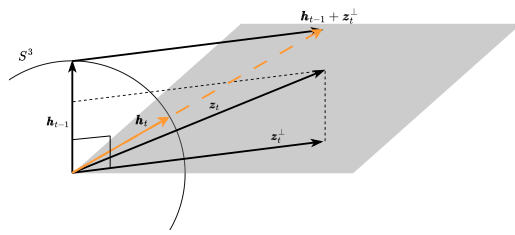


Figure 5: A single update with the tangent recurrent unit in $\mathbb{R}^3$.

Let $\boldsymbol{B} \in \mathbb{R}^{d \times k}$ where $\boldsymbol{B}_{ij} \sim \mathcal{N}(0,1)$ and $b_j = ||\boldsymbol{B}_{*j}||_2$. Assume that $\boldsymbol{h}_0$ is initialized such that $||\boldsymbol{h}_0||_2 = 1$. Define $\tilde{\boldsymbol{B}} = \frac{1}{\sqrt{k}} \left[\frac{\boldsymbol{B}_{*1}}{b_1} \mid \cdots \mid \frac{\boldsymbol{B}_{*k}}{b_k} \right]$ (each column of $\boldsymbol{B}$ is normalized so that $||\tilde{\boldsymbol{B}}_{*j}||_2 = 1/\sqrt{k}$). We do this to ensure that the random input transformation weighs individual features by the same amount (since $\boldsymbol{z}_t = \sum_{j=0}^{d} (\boldsymbol{x}_t)_j \tilde{\boldsymbol{B}}_{*j}$).

7.1 TRU fixed point

Theorem 1. *Let $\boldsymbol{x} \in \mathbb{R}^k$ be an input with $||\boldsymbol{x}||_2 = 1$, and define $\boldsymbol{z} := \tilde{\boldsymbol{B}}\boldsymbol{x} \in \mathbb{R}^d$. If $||\boldsymbol{z}||_2 > 0$, and if*

$$\boldsymbol{h}_+^* = \frac{\boldsymbol{z}}{||\boldsymbol{z}||_2} \quad \textit{and} \quad \boldsymbol{h}_-^* = -\boldsymbol{h}_+^*,$$

then repeated application of f_θ reaches the fixed point $\boldsymbol{h}_+^*$ for all $\boldsymbol{x}$ and all initializations of $\boldsymbol{h}_0 \neq \boldsymbol{h}_-^*$:

$$f_\theta(\boldsymbol{h}_+^*, \boldsymbol{x}) = \boldsymbol{h}_+^*.$$

Proof. We need to show two things. First, that $\boldsymbol{h}_+^*$ and $\boldsymbol{h}_-^*$ are fixed points. Second, that $\boldsymbol{h}_+^*$ attracts every initialization $\boldsymbol{h}_0 \neq \boldsymbol{h}_-^*$. We start with some preliminaries.

Since $\boldsymbol{x}$ and $\tilde{\boldsymbol{B}}$ does not change at every step, $\boldsymbol{z}$ is also fixed. Define $p(\boldsymbol{h}) := (\boldsymbol{I} - \boldsymbol{h}\boldsymbol{h}^\intercal)\boldsymbol{z}$, and note that $p(\boldsymbol{h})$ is a projection of $\boldsymbol{z}$ to the orthogonal complement of $\boldsymbol{h}$. The TRU operation over fixed inputs is then

$$F_x(\boldsymbol{h}) = f_\theta(\boldsymbol{h}, \boldsymbol{x}) = \frac{\boldsymbol{h} + p(\boldsymbol{h})}{||\boldsymbol{h} + p(\boldsymbol{h})||_2}. \tag{5}$$

Additionally, since $\tilde{\boldsymbol{B}}$ is column-normalized with $||\tilde{\boldsymbol{B}}_{*j}||_2 = 1/\sqrt{k}$, we know that

$$||\tilde{\boldsymbol{B}}||_F^2 = \sum_{i=1}^{k} ||\tilde{\boldsymbol{B}}_{*j}||_2 = k \cdot \frac{1}{k} = 1.$$

Since the Frobenius matrix norm is larger than the spectral norm,

$$||\boldsymbol{z}||_2 = ||\tilde{\boldsymbol{B}}\boldsymbol{x}||_2 \leq ||\tilde{\boldsymbol{B}}||_2 ||\boldsymbol{x}||_2 \leq ||\tilde{\boldsymbol{B}}||_F ||\boldsymbol{x}||_2 = 1$$

So $0 < ||\boldsymbol{z}||_2 \leq 1$.

Now we show that $\boldsymbol{h}_+^*$ is one of two only fixed points. For a fixed point for F_x, at every step the denominator in Equation 5 only scales the numerator $\boldsymbol{h} + p(\boldsymbol{h})$. This means that for *every* fixed point,

$\boldsymbol{h} + p(\boldsymbol{h})$ must be a scalar multiple of $\boldsymbol{h}$. A fixed point satisfies that $F_x(\boldsymbol{h}) = \boldsymbol{h}$. This means that $p(\boldsymbol{h}) = 0$, and

$$(\boldsymbol{I} - \boldsymbol{h}\boldsymbol{h}^\intercal)\boldsymbol{z} = \boldsymbol{z} - \boldsymbol{h}\boldsymbol{h}^\intercal\boldsymbol{z} = 0 \Rightarrow (\boldsymbol{h}^\intercal\boldsymbol{z})\boldsymbol{h} = \boldsymbol{z},$$

which means that a fixed point satisfies that $\boldsymbol{h}$ is parallel to $\boldsymbol{z}$. Since $||\boldsymbol{h}||_2 = 1$, this happens exactly when

$$\boldsymbol{h} = \pm\frac{\boldsymbol{z}}{||\boldsymbol{z}||_2}$$

which shows that $\boldsymbol{h}_+^*$ and $\boldsymbol{h}_-^*$ are the only fixed points.

Now we show that $\boldsymbol{h}_+^*$ is an attractor. Let us consider the angle between $\boldsymbol{h}_+^*$ and $\boldsymbol{h}_t$ at every time step t. Since all recurrent states have $||\boldsymbol{h}||_2 = 1$, if $\theta_t \in [0, \pi]$ is the angle between $\boldsymbol{h}_t$ and $\boldsymbol{h}_+^*$, $\boldsymbol{h}_t^\intercal\boldsymbol{h}_+^* = \cos\theta_t$. This means the orthogonal projection of $\boldsymbol{h}_+^*$ onto $\boldsymbol{h}_t$ is $(\boldsymbol{h}_t^\intercal\boldsymbol{h}_+^*)\boldsymbol{h}_t = \cos\theta_t\boldsymbol{h}_t$. The vector from $\cos\theta_t\boldsymbol{h}_t$ to $\boldsymbol{h}_+^*$ is then $\boldsymbol{h}_+^* - \cos\theta_t\boldsymbol{h}_t$. The size of this vector is:

$$||\boldsymbol{h}_+^* - \cos\theta_t\boldsymbol{h}_t||_2 = \sqrt{1 - \cos^2\theta_t} = \sin\theta_t.$$

So normalizing $\boldsymbol{h}_+^* - \cos\theta_t\boldsymbol{h}_t$ to a unit vector gives

$$\boldsymbol{e}_t = \frac{\boldsymbol{h}_+^* - \cos\theta_t\boldsymbol{h}_t}{\sin\theta_t} \Rightarrow \boldsymbol{h}_+^* = \cos\theta_t\boldsymbol{h}_t + \sin\theta_t\boldsymbol{e}_t$$

which decomposes $\boldsymbol{h}_+^*$ into its components $\boldsymbol{h}_t$ and $\boldsymbol{e}_t$. We can see this decomposition in Figure 6.

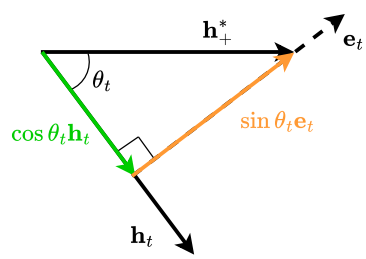


Figure 6: Geometry of $\boldsymbol{h}_+^*$, $\boldsymbol{h}_t$, and $\boldsymbol{e}_t$.

Using this decomposition, we can consider $p(\boldsymbol{h}_t)$:

$$\begin{aligned} p(\boldsymbol{h}_t) &= (\boldsymbol{I} - \boldsymbol{h}_t\boldsymbol{h}_t^\intercal)\boldsymbol{z} \\ &= (\boldsymbol{I} - \boldsymbol{h}_t\boldsymbol{h}_t^\intercal)||\boldsymbol{z}||_2\boldsymbol{h}_+^* \\ &= ||\boldsymbol{z}||_2(\boldsymbol{h}_+^* - (\boldsymbol{h}_t^\intercal\boldsymbol{h}_+^*)\boldsymbol{h}_t) \end{aligned}$$

Since $\boldsymbol{h}_+^* = \cos\theta_t\boldsymbol{h}_t + \sin\theta_t\boldsymbol{e}_t$ and $(\boldsymbol{h}_t^\intercal\boldsymbol{h}_+^*)\boldsymbol{h}_t = \cos\theta_t\boldsymbol{h}_t$,

$$\begin{aligned} p(\boldsymbol{h}_t) &= ||\boldsymbol{z}||_2(\cos\theta_t\boldsymbol{h}_t + \sin\theta_t\boldsymbol{e}_t - \cos\theta_t\boldsymbol{h}_t) \\ &= ||\boldsymbol{z}||_2\sin\theta_t\boldsymbol{e}_t. \end{aligned}$$

Let us define $a_t = ||\boldsymbol{z}||_2\sin\theta_t$. The unnormalized update gives:

$$\hat{\boldsymbol{h}}_{t+1} = \boldsymbol{h}_t + p(\boldsymbol{h}_t) = \boldsymbol{h}_t + a_t\boldsymbol{e}_t.$$

The normalization term is

$$||\hat{\boldsymbol{h}}_{t+1}||_2 = \sqrt{1 + a_t^2}.$$

The normalized update is

$$\boldsymbol{h}_{t+1} = \frac{1}{\sqrt{1 + a_t^2}}\boldsymbol{h}_t + \frac{a_t}{\sqrt{1 + a_t^2}}\boldsymbol{e}_t.$$

By trigonometric rules, we can define $\phi_t = \arctan(a_t) = \arctan(||\boldsymbol{z}||_2 \sin\theta_t)$ giving us

$$\boldsymbol{h}_{t+1} = \cos\phi_t \boldsymbol{h}_t + \sin\phi_t \boldsymbol{e}_t.$$

This implies that $\boldsymbol{h}_{t+1}$ is obtained by rotating $\boldsymbol{h}_t$ towards $\boldsymbol{h}_+^*$ by ϕ_t. Since θ_t is the total angle between $\boldsymbol{h}_t$ and $\boldsymbol{h}_+^*$, we want to be able to write $\theta_{t+1} = \theta_t - \phi_t$. To do so, we need to show that $\phi_t \leq \theta_t$. If $\theta_t \leq \pi/2$,

$$||\boldsymbol{z}||_2 \sin\theta_t \leq \sin\theta_t \leq \tan\theta_t \Rightarrow \arctan(||\boldsymbol{z}||_2 \sin\theta_t) \leq \theta_t \Rightarrow \phi_t \leq \theta_t.$$

And for $\theta_t > \pi/2$, since $\phi_t = \arctan(||\boldsymbol{z}||_2 \sin\theta_t)$, $\phi_t \leq \pi/2 < \theta_t$. This gives us

$$\theta_{t+1} = \theta_t - \phi_t = \theta_t - \arctan(||\boldsymbol{z}||_2 \sin\theta_t).$$

Since $\theta_t \in (0, \pi)$, $\sin\theta_t > 0$ and $||\boldsymbol{z}||_2 > 0$, then $\arctan(||\boldsymbol{z}||_2 \sin\theta_t) > 0$ and $\theta_{t+1} < \theta_t$. Thus θ_t decreases monotonically and

$$\lim_{t\to\infty} \theta_t = 0.$$

□

8 Experimental details

We list all experimental details below, including environment details and hyperparameters tuned.

8.1 Environment details

We provide environment and training hyperparameters in Table 1. We now describe each environment in detail.

Clock. The Clock environment is a fixed-horizon episodic POMDP on a circular number line with positions $\{1, \ldots, 5\}$. The agent chooses to increment or decrement its position, with transitions wrapping around ($5 + 1 = 1$ and $1 - 1 = 5$). With probability $\epsilon = 0.2$, the transition is overridden by a uniform random jump. Reward is given only at termination: $+1$ if the cumulative sum of observations is divisible by 4, and $+10$ if divisible by 7 (stacking to $+11$ when both hold). Because the reward depends on the running sum of the entire sequence, the agent must explore over trajectories to discover reward-yielding patterns.

RockSample(11, 11). RockSample (Smith & Simmons, 2004) places an agent on an 11×11 grid containing 11 rocks, each independently good or bad with equal probability. The agent has 4 cardinal movement actions, 1 sample action ($+10$ for a good rock, -10 for a bad rock), and 11 check actions (one per rock). Checking rock i returns a noisy binary signal with accuracy $\frac{1}{2}(1 + 2^{-d/d_{\text{half}}})$, where d is the distance to rock i and $d_{\text{half}} = 20$. The agent receives $+10$ for exiting through the right edge of the grid. The observation encodes the agent's position via two one-hot vectors, the most recent check result for each rock (-1 bad, 0 unchecked, $+1$ good), and the previous action.

PointMaze Visual. PointMaze (Fu et al., 2020; Radji, 2025) is a continuous-control navigation task in the Giant_Diverse_G maze layout. The agent controls a point mass via 2D forces in $[-1, 1]^2$ and must reach a goal sampled from predefined positions. The observation is an egocentric top-down binary image showing walls within a fixed radius, flattened and concatenated with the previous action. The agent receives $+1$ upon reaching the goal. Since a single frame reveals only local wall structure, the agent must integrate observations over time to localize.

My Way Home. My Way Home is a first-person visual navigation task from ViZDoom (Wydmuch et al., 2019), implemented in JAXenstein (Tao & Konidaris, 2026) for hardware-accelerated training. The agent spawns at a random location in a maze and must navigate to a fixed goal room for a reward of $+1$. Observations are $64 \times 64 \times 3$ RGB images from the agent's viewpoint. The large maze, random spawning, and purely visual observations require the agent to systematically explore different trajectories to discover the goal.

Table 1: Environment and fixed training details for the PPO RND exploration experiments.

Environment	Observation / action / horizon	Steps	Num. Envs	Rollout	PPO hid.	RND hid.	γ	λ_{GAE}
Clock	Scalar lock state plus previous one-hot action; 3 input features. Discrete 2-action control. Horizon 20.	2×10^7	8	128	256	256	0.99	0.7
RockSample(11, 11)	Position features and rock-check signal plus previous one-hot action; 49 input features. Discrete 16-action control. Horizon 1000.	5×10^6	16	128	256	256	0.99	0.9
PointMaze Visual	Flattened 20×20 local image plus previous continuous action; 402 input features. Continuous 2D action in $[-1,1]^2$. Horizon 2000.	10^7	16	128	256	128	0.999	0.95
My Way Home	$64\times64\times3$ RGB observation; previous discrete action provided separately as a one-hot vector of length 5. Discrete 5-action control. Horizon 2100.	5×10^7	128	128	256	128	0.999	0.95

Table 2: PPO hyperparameter sweeps. Singleton sets indicate values passed through the sweep interface but not varied within that environment.

Environment	Learning rate	λ_{GAE}	Entropy coefficient
Clock	$\{2.5\times10^{-4}\}$	$\{0.7\}$	$\{0.001, 0.01, 0.05\}$
RockSample(11, 11)	$\{2.5\times10^{-3}\}$	$\{0.9\}$	$\{0.001, 0.01, 0.05\}$
PointMaze Visual	$\{2.5\times10^{-4}\}$	$\{0.95\}$	$\{0.01\}$
My Way Home	$\{2.5\times10^{-4}\}$	$\{0.95\}$	$\{0.01\}$

All PPO experiments were conducted over 5 seeds, except for My Way Home, which was conducted over 3 seeds. Hyperparameters that were not swept were chosen either from default PPO hyperparameters (Lu et al., 2022), or from previous experiments where base PPO worked best. RND (Burda et al., 2019) computes an intrinsic reward as the prediction error between a frozen target network and a learned predictor network. Both networks share the same hidden size (Table 1). The three variants differ in the memory architecture of the target and predictor:

(1) **RND-FC**: Both the predictor and target are feedforward networks with no recurrence (`memory_cell=fc`).

(2) **RND-GRU**: The predictor is a learned GRU; the target is a frozen, randomly initialized GRU.

(3) **RND-TRU**: The predictor is a learned GRU; the target is a frozen TRU with an identity Q-matrix. Timescale parameters are set to $\alpha_{\min} = 1.0$ and $\alpha_{\max\text{-scale}} = 1.0$.

We sweep the intrinsic reward coefficient and the RND loss coefficient per environment as shown in Table 3.

Table 3: RND hyperparameter sweeps for RND-FC, RND-GRU and RND-TRU. All environments use normalized RND inputs and $\gamma_{\text{RND}} = 0.99$.

Environment	RND reward coef.	RND loss coef.
Clock	$\{0.1, 1.0, 10.0\}$	$\{1, 10, 100\}$
RockSample(11, 11)	$\{0.01, 0.1, 1.0, 10.0\}$	$\{1, 10, 100, 1000\}$
PointMaze Visual	$\{0.1, 1.0, 10.0\}$	$\{1, 10, 100\}$
My Way Home	$\{0.1, 1.0, 10.0\}$	$\{0.01, 0.1\}$

8.2 Intrinsic curiosity module details

As a baseline we implement a recurrent version of the intrinsic curiosity module (Pathak et al., 2017). The ICM module consists of three components: a recurrent encoder, a forward model, and an inverse model. The recurrent encoder maps observation sequences through a 2-layer MLP followed by a GRU with a 128-dimensional hidden state, producing embedding features $\boldsymbol{\phi}_t \in \mathbb{R}^{128}$ at each time step. The forward model takes the current embedding and action as input and predicts the next embedding through a 2-layer MLP. The inverse model concatenates consecutive embedding and predicts the intervening action through a 2-layer MLP.

The intrinsic reward is the mean squared prediction error of the forward model in latent space.

$$r_t^{\text{int}} = \lambda_{\text{ICM}} \cdot \frac{1}{d} \|\boldsymbol{\phi}_{t+1} - \hat{\boldsymbol{\phi}}_{t+1}\|_2^2$$

where $\hat{\boldsymbol{\phi}}_{t+1}$ is the forward model prediction, d is the latent dimension, and λ_{ICM} is the reward coefficient. The intrinsic reward is set to zero at terminals and added to the extrinsic reward with a stop-gradient, so ICM gradients do not backpropagate into the actor or critic. The ICM networks are trained with a separate Adam optimizer. During training, full sequences of length `seq_len` are sampled from the replay buffer, and the recurrent encoder processes them with reset masks at episode boundaries to maintain proper temporal context. Unlike the CFN variants, no running-variance normalization is applied to the intrinsic reward.

Table 4: ICM hyperparameter sweeps. The ICM learning rate, forward-loss coefficient, and inverse-loss coefficient are the same across environments.

Environment	Reward coef.	Forward coef.	Inverse coef.	ICM LR
Clock	$\{0.1, 1.0, 10.0\}$	$\{0.2\}$	$\{0.8\}$	$\{3\times10^{-4}\}$
RockSample(11, 11)	$\{0.01, 0.1, 1.0, 10.0\}$	$\{0.2\}$	$\{0.8\}$	$\{3\times10^{-4}\}$
PointMaze Visual	$\{0.01, 0.1, 1.0, 10.0\}$	$\{0.2\}$	$\{0.8\}$	$\{3\times10^{-4}\}$
My Way Home	$\{0.1, 1.0, 10.0\}$	$\{0.2\}$	$\{0.8\}$	$\{3\times10^{-4}\}$

8.3 Soft actor-critic details

We use a recurrent variant of soft actor-critic (SAC) (Haarnoja et al., 2018) with GRU-based actor and critic networks. Sequences of fixed length are stored in a replay buffer along with the GRU hidden state at the start of each chunk, allowing proper recurrence during training. Temperature is automatically tuned via dual gradient descent on the target entropy. Clock and RockSample use $\gamma = 0.99$ with 1-step returns, while PointMaze uses $\gamma = 0.999$ with 50-step returns. We sweep the initial temperature and target entropy per environment. All swept hyperparameters are listed in Table 5.

Table 5: SAC fixed training hyperparameters. All environments use $\tau = 0.005$, batch size 256, hidden size 256, 16 parallel environments, and 10 gradient updates per step.

Environment	LR	Steps	Buffer size	Learn starts	Seq len	Init temp.	Target entropy
Clock	2.5×10^{-4}	10^7	2×10^5	10^4	128	$\{0.1, 0.5, 1.0\}$	$\{0.1, 0.3, 0.5\}$
RockSample	2.5×10^{-3}	5×10^6	2×10^5	5×10^4	128	$\{0.1, 0.5, 1.0\}$	$\{0.5, 1.0, 2.0\}$
PointMaze	2.5×10^{-4}	10^7	5×10^5	5×10^4	256	$\{0.1, 0.5, 1.0\}$	$\{1.0, 2.0, 3.0\}$

SAC Performance on PointMaze PointMaze Visual is a particularly challenging environment for SAC-based methods due to the combination of a long horizon (maximum episode 2000 steps) and extremely sparse rewards (a single $+1$ upon reaching the goal). Even with $\gamma = 0.999$ and 50-step returns, the effective discount over a full episode is $0.999^{2000} \approx 0.135$, meaning that even successful trajectories yield very sparse discounted returns. This can lead to significantly lower performance

on PointMaze when comparing SAC-based and PPO-based methods. Also, in such sparse-reward settings, the intrinsic CFN reward can easily dominate the extrinsic signal, making the choice of reward scale particularly important.

8.4 Coin flipping network details

Coin flipping networks (CFN) (Lobel et al., 2023) approximate pseudocounts by training a predictor network to match random ± 1 coin flip targets associated with each visited state. The prediction error serves as a density-based exploration bonus. Frequently visited states are easier to predict, yielding smaller bonuses. We integrate CFN with SAC using three memory variants that differ in how the CFN input features are computed.

(1) **CFN-FC**: The predictor and a frozen prior network are feedforward MLPs that take the current observation as input. This variant only explores over observations.

(2) **CFN-GRU**: A frozen, randomly initialized `FixedRecurrentEncoder` processes the observation-action history. The encoder maps observations through a linear embedding layer followed by a frozen GRU, producing a fixed-length hidden state at each time step. The predictor and prior networks then operate on this recurrent hidden state rather than the raw observation. This enables exploration over trajectories.

(3) **CFN-TRU**: Same architecture as CFN-GRU, but the frozen recurrence is a tangent recurrent unit (Section 4.3) with an identity Q-matrix and timescale parameters $\alpha_{\min} = 1.0$, $\alpha_{\text{max-scale}} = 1.0$.

For all variants, the predictor and prior are 2-hidden-layer MLPs with ReLU activations and hidden size equal to `cfn_hidden_size`. The intrinsic reward at each step is computed as $r_t^{\text{int}} = \lambda_{\text{CFN}} \cdot \frac{m_t - \hat{\mu}_m}{\sqrt{\hat{\sigma}_m^2 + \epsilon}}$, where $m_t = \left(\frac{1}{d}\|f_{\text{pred}}(\phi_t) + \bar{f}_{\text{prior}}(\phi_t)\|_2^2\right)^{0.5}$ is the magnitude of the combined trained and normalized-prior predictions, and $\hat{\mu}_m$, $\hat{\sigma}_m^2$ are running statistics used to normalize the reward. Here ϕ_t denotes either the raw observation (CFN-FC) or the frozen recurrent hidden state (CFN-GRU, CFN-TRU). The CFN predictor is trained with MSE loss against the coin flip targets using a separate Adam optimizer. Coin flips are resampled every step ($p_{\text{replace}} = 1.0$). The recurrent encoder hidden state is stored at sequence boundaries in the replay buffer so that recurrence can be properly resumed during training. For all recurrent CFN variants, observation inputs to the frozen encoder are normalized using running statistics.

Table 6: CFN hyperparameter sweeps for CFN-FC, CFN-GRU, and CFN-TRU. All variants use a bonus exponent of 0.5.

Environment	CFN LR	Reward scale	Output dim	CFN batch size
Clock	2.5×10^{-4}	$\{0.01, 0.1, 1.0\}$	20	512
RockSample	2.5×10^{-3}	$\{0.01, 0.1, 1.0\}$	20	512
PointMaze	2.5×10^{-4}	$\{10^{-3}, 10^{-2}, 10^{-1}\}$	100	256

8.5 Best Hyperparameters

Table 7: Best swept hyperparameters per algorithm and environment for SAC based methods.

Environment	**Algorithm**	**Init Temperature**	**Target Entropy**	**Reward Scale**
Clock	SAC	0.1	0.1	–
	SAC-ICM	0.1	0.1	1.0
	CFN-FC	0.1	0.1	0.01
	CFN-GRU	0.1	0.1	0.01
	CFN-TRU	0.1	0.1	0.1
RockSample	SAC	1.0	2.0	–
	SAC-ICM	1.0	2.0	0.1
	CFN-FC	0.5	1.0	0.01
	CFN-GRU	0.5	2.0	0.01
	CFN-TRU	1.0	2.0	0.01
PointMaze	SAC	1.0	-2.0	–
	SAC-ICM	0.1	-2.0	1.0
	CFN-FC	1.0	1.0	10^{-1}
	CFN-GRU	0.1	1.0	10^{-2}
	CFN-TRU	0.5	1.0	10^{-1}

Table 8: Best swept hyperparameters per algorithm and environment for PPO-based methods.

Environment	Algorithm	LR	Ent, coef.	GAE λ	Intrinsic coef.	RND loss
Lock	PPO	2.5×10^{-4}	10^{-2}	0.70	–	–
	PPO-ICM	2.5×10^{-4}	10^{-2}	0.70	10	–
	RND-FC	2.5×10^{-4}	10^{-2}	0.70	10	100
	RND-GRU	2.5×10^{-4}	10^{-2}	0.70	1	10
	RND-TRU	2.5×10^{-4}	10^{-2}	0.70	0.1	100
RockSample	PPO	2.5×10^{-3}	10^{-2}	0.90	–	–
	PPO-ICM	2.5×10^{-3}	5×10^{-2}	0.90	0.1	–
	RND-FC	2.5×10^{-3}	10^{-2}	0.90	0.1	1
	RND-GRU	2.5×10^{-3}	5×10^{-2}	0.90	0.1	10
	RND-TRU	2.5×10^{-3}	10^{-2}	0.90	10^{-2}	100
PointMaze Visual	PPO	2.5×10^{-4}	5×10^{-2}	0.95	–	–
	PPO-ICM	2.5×10^{-4}	10^{-3}	0.95	0.1	–
	RND-FC	2.5×10^{-4}	10^{-3}	0.95	10^{-2}	1
	RND-GRU	2.5×10^{-4}	10^{-3}	0.95	10	1
	RND-TRU	2.5×10^{-4}	10^{-2}	0.95	1	100
My Way Home	PPO	2.5×10^{-4}	10^{-2}	0.95	–	–
	PPO-ICM	2.5×10^{-4}	10^{-2}	0.95	1	–
	RND-FC	2.5×10^{-4}	10^{-2}	0.95	10	10^{-2}
	RND-GRU	2.5×10^{-4}	10^{-2}	0.95	0.1	0.1
	RND-TRU	2.5×10^{-4}	10^{-2}	0.95	0.1	0.1