Forager: a lightweight testbed for continual learning with partial observability in reinforcement learning

Steven Tang, Xinze Xiong, Anna Hakhverdyan, Andrew Patterson, Jacob Adkins, Jiamin He, Esraa Elelimy, Parham Mohammad Panahi, Martha White, Adam White

Keywords: reinforcement learning, continual reinforcement learning, never–ending learning, environments, lifelong learning, benchmarks, partially observable

Summary

In continual reinforcement learning (CRL), good performance requires never-ending learning, acting, and exploration in a big, partially observable world. Most CRL experiments have focused on loss of plasticity—the inability to keep learning—in one-off experiments where some unobservable non-stationarity is added to classic fully observable MDPs. Further, these experiments rarely consider the role of partial observability and the importance of CRL agents that use memory or recurrence. One potential reason for this focus on mitigating loss of plasticity without considering partial observability is that many partially-observable CRL environments are prohibitively expensive. In this paper, we introduce Forager, a lightweight partially-observable CRL environment with a constant memory footprint. We provide a set of experiments and sample tasks demonstrating that Forager is challenging for current CRL agents and yet also allows for in-depth study of those agents. We demonstrate that agents exhibit loss of plasticity, proposed mitigations can help, but that most useful is to leverage state construction. We conclude with a variant of Forager that generates an unending stream of new tasks to learn that clearly highlights the limitations of current CRL agents.

Contribution(s)

1. A new testbed for continual learning research that is resource efficient allowing for fast long-running experiments and rapid experiment iteration. Forager is configurable with user-specified parameters that allow fine control of the degree of partial observability and problem difficulty. We provide several sample use cases, including one that uses random generation to create an unending stream of new tasks to ensure that agents must continue to learn forever and never converge as highlighted as a key feature of continual learning in the literature (Abel et al., 2023). This is all wrapped up in an open-source repository.
   Context: Jelly Bean World (Platanios et al., 2020) was designed under similar principles, but JBW has technical issues in terms of memory usage demonstrated in the introduction. Agar.io (Mohamed et al., 2026) is another related benchmark, but successful learning has not been demonstrated in this domain. Forager is not realistic and the action and observation space are simple. Forager has no physics, nor other features of rich game-like environments. This is by design. There are plenty of complex environments available to RL researchers. The goal here is to balance rapid prototyping of ideas with just the right level of complexity to challenge current algorithms.
2. We highlight and demonstrate the role of state construction in continual RL. We show that feed forward agents exhibit loss of plasticity which can be mitigated to some degree by recent algorithms. However, simple memory traces and recurrent PPO agents can both mitigate some partial observability and exhibit continual learning and tracking. This highlights the need for new recurrent CRL algorithms.
   Context: Recent work demonstrated that the recurrent PPO achieves reliable Foraging in Craftax (Simmons-Edler et al., 2025), but this required access to privileged state information and many parallel asynchronous actors. Here we focus on continual learning and state construction in the single stream setting as it remains a key challenge in CRL (Abel et al., 2023; Khetarpal et al., 2022) and asynchronous training can mask this challenge (Mayor et al., 2025).

Forager: a lightweight testbed for continual learning with partial observability in reinforcement learning

Steven Tang[1,2], **Xinze Xiong**[1,2], **Anna Hakhverdyan**[1,2], **Andrew Patterson**[1,2], **Jacob Adkins**[1,2], **Jiamin He**[1,2], **Esraa Elelimy**[1,2], **Parham Mohammad Panahi**[1,2], **Martha White**[1,2,3], **Adam White**[1,2,3]

{stang5, xinze5, hakhverd, ap3, jadkins, jiamin12, elelimy, parham1, whitem, amw8}@ualberta.ca

[1]**Department of Computing Science, University of Alberta, Canada**
[2]**Alberta Machine Intelligence Institute (Amii)**
[3]**CIFAR AI Chair**

Abstract

In continual reinforcement learning (CRL), good performance requires never-ending learning, acting, and exploration in a big, partially observable world. Most CRL experiments have focused on loss of plasticity—the inability to keep learning—in one-off experiments where some unobservable non-stationarity is added to classic fully observable MDPs. Further, these experiments rarely consider the role of partial observability and the importance of CRL agents that use memory or recurrence. One potential reason for this focus on mitigating loss of plasticity without considering partial observability is that many partially-observable CRL environments are prohibitively expensive. In this paper, we introduce Forager, a lightweight partially-observable CRL environment with a constant memory footprint. We provide a set of experiments and sample tasks demonstrating that Forager is challenging for current CRL agents and yet also allows for in-depth study of those agents. We demonstrate that agents exhibit loss of plasticity, proposed mitigations can help, but that most useful is to leverage state construction. We conclude with a variant of Forager that generates an unending stream of new tasks to learn that clearly highlights the limitations of current CRL agents.

1 Introduction

Continual Reinforcement Learning (CRL) is the study of agents that must continue to learn forever in order to perform well. In the real world, a person or an agent has a limited way in which it observes the world: it can only see, hear, and touch things in its immediate vicinity—the world is partially observable. In addition, there are effectively an infinite number of things to learn about. This view-point, called the *Big World Hypothesis* (Javed & Sutton, 2024), is often summarized as the agent being much smaller than the environment. In such situations, the agent cannot represent the optimal policy and any world model will be approximate. The world is partially observable and from the agent's perspective often appears non-stationary. The best approach is to construct an internal state to mitigate partial observability (e.g., recurrent architectures) and to track (Sutton et al., 2007) to learn continually in deployment (Abbas et al., 2023; Dohare et al., 2024).

Most work in reinforcement learning (RL) focuses on fully observable benchmarks where the agent is much larger than the environment. For example, the Arcade Learning Environment (ALE) (Bellemare et al., 2013) requires only 128 bytes of RAM and 4KB of ROM, whereas Nature DQN (a small agent by today's standards) contains over 1.7 million learnable parameters (Mnih et al., 2015). In addition, frame-stacking largely mitigates partial observability in ALE, while other benchmarks,

like Mujoco, give the agent direct access to MDP state. Larger partially observable environments exist, such as Minecraft (Guss et al., 2021), but many of these require complex agent architectures and significant compute, even when considering lightweight versions like Craftax (Matthews et al., 2024). Consequently, isolating research questions and incremental algorithmic development becomes challenging. As a result, many CRL papers introduce a new toy CRL task, usually by taking a known environment and introducing some non-stationarity, and then conclude with a separate set of experiments in fully observable benchmarks (Lee et al., 2024; 2023; Lyle et al., 2023; 2024a;b; Sokar et al., 2023; Nikishin et al., 2023; Lyle et al., 2022; Elsayed & Mahmood, 2024).

Work in CRL has largely focused on maintaining network trainability in the face of task non-stationarity but does not emphasize the role of partial observability. RL agents can both catastrophically forget and lose all ability to learn (so called loss of plasticity) (Abbas et al., 2023; Dohare et al., 2024; Anand & Precup, 2023; Lyle et al., 2024b) when faced with either a sequence of tasks or some unobservable source of non-stationarity. Although loss of plasticity is widely observed, the precise cause remains elusive and a wide variety of potential culprits have been put forward including: dead and dormant neurons (Abbas et al., 2023; Sokar et al., 2023), the beneficial properties of network initializations eroding over time (Kumar et al., 2020; Lewandowski et al., 2025; Galashov et al., 2024; Dohare et al., 2024; Anand & Precup, 2023), changes in target magnitude (Lyle et al., 2024b), and even ineffective hyper-parameter tuning (Mesbahi et al., 2025). The solutions, various forms of regularization, resetting, normalization, and special network architectures, all focus on maintaining plasticity, ensuring the agent can continually relearn in response to non-stationarity induced by task switches. Although task switching has been noted as a form of partial observability (Khetarpal et al., 2022), to the best of our knowledge, prior work has focused on methods that enable continual learning of the policy and value function, but state-construction remains underexplored in CRL.

In this paper we introduce a new testbed and set of experiments exploring an agent's ability to both track (continually update the value function and policy) and construct state. We introduce a lightweight environment generation testbed for continual learning called Forager. The agent's objective is to collect and avoid objects in a gridworld based on a limited, overhead field of view. Forager can run up to a hundred thousand frames per second with a constant memory footprint (as shown in Figure 1) giving it a significant advantage over other CRL benchmarks like Jelly Bean World (see Section 2 for a discussion of related testbeds). Forager is parameterized by an adjustable field of view and world size, allowing careful control of the degree of partial observability. Forager is available in two implementations. The CPU implementation allows running long-running experiments very quickly and is available at https://github.com/steventango/forager. The JAX implementation, Foragax, allows for massive parallelism for hundreds of independent trials in parallel on GPU and is available at https://github.com/steventango/continual-foragax. Foragax is useful for obtaining hyper-parameter sweeps and running a sufficient number of independent trials to achieve statistical significance. Finally, we specify several test cases as a starting point, including a variant that produces an infinite series of continual learning tasks that require both tracking and continual state construction.

Our experiments reveal, unsurprisingly, that popular deep RL agents fail to learn continually or settle on suboptimal, static foraging policies. More interestingly, recently proposed continual learning approaches like regularization (Kumar et al., 2024; Ash & Adams, 2020), CReLU activations (Abbas et al., 2023), and using multiple networks (Anand & Precup, 2023) do not help resolve the partial observability in Forager. Some methods appear to address loss of plasticity, however, they show limited success in continual learning through tracking. Whereas state construction methods, like simple memory traces (Rafiee et al., 2023; Janjua et al., 2023) and recurrent architectures (Elelimy et al., 2024) enable tracking but are still suboptimal compared to search baselines. The Forager benchmark suite and our extensive set of results provide a clear foundation for investigating continual learning in partially observable environments where both continual state construction and tracking are required, just like in the real world.

2 Related CRL testbeds and benchmarks

Currently available RL benchmarks are not well-suited to developing new CRL algorithms. Many of our large-scale benchmarks are not big worlds. The ALE environment is actually an MDP under-the-hood, which a recurrent agent should be able to perfectly simulate. The same is true of Mujoco, DM Control, and DMLab. There are several large-scale, realistic environments but these require long learning times and prohibitive computation, such as Minecraft (Tessler et al., 2017; Hafner et al., 2025) and Causal World (Ahmed et al., 2021; Sharma et al., 2022). NetHack (Küttler et al., 2020) and Crafter (Hafner, 2022) are more computationally frugal variants, but remain quite complex and slow. Another approach is to take already expensive environments and switch between them, such as in Switching ALE (Abbas et al., 2023), Cora (Powers et al., 2022), COOM based on Visual Doom tasks (Tomilin et al., 2023), and switching between Minigrid tasks (Anand & Precup, 2023). These benchmarks are computationally challenging as CRL experiments already necessitate longer training time to demonstrate successful learning or loss of plasticity.

Craftax (Matthews et al., 2024), substantially improves on Crafter but becomes impractical when execution is restricted to a single-stream of experience. Craftax was recently used to benchmark foraging in an environment with food, water, and enemies (Simmons-Edler et al., 2025). Interestingly, recurrent PPO was found to be highly effective, but required both significant environment parallelism and access to the privileged global (x, y) position of the agent via an auxiliary loss target. Although asynchronous training is widely used, support for single-stream RL is important for algorithmic progress (Abel et al., 2023; Mesbahi et al., 2025). In addition, recent work has shown that the use of asynchronous training mitigates loss of plasticity (Mayor et al., 2025), effectively masking one of the key challenges of CRL.

Many papers introduce a bespoke, lightweight environment to highlight the challenges of continual learning. These environments are typically variants of supervised learning benchmarks or created by adding non-stationarity or task-switching to an existing RL environment. Examples include Up n Down (Nikishin et al., 2023), image-classification MDPs (Lyle et al., 2023), and Slippery Ant (Dohare et al., 2024). In many cases, these tasks are never used again in followup work.

There are some environments designed to be big worlds. In Jelly Bean World (JBW), the agent navigates an infinite, procedurally generated gridworld collecting and avoiding objects. The agent's observation, like Forager, is an agent-centric, limited top-down field of view which, in principle, requires recurrent architectures or some other form of memory. Indeed, JBW was one of the few testbeds highlighted in Khetarpal et al. (2022)'s survey. Yet, to the best of our knowledge, only two papers have used JBW (Anand & Precup, 2023; Mesbahi et al., 2025). One potential explanation is that JBW runs relatively slowly and has a memory footprint that grows as the agent explores, as demonstrated in Figure 1. In contrast, the Forager CPU implementation is significantly faster and the Forager JAX implementation, Foragax, enables running hundreds of independent trials in parallel on a single GPU. The better the agent becomes at continual learning, the more memory JBW uses and the slower it runs. As we discuss in the Supplement (Sections 12 & 13), Forager does not use procedural generation, but still produces quantitatively similar results to JBW with a fixed memory footprint at a fraction of the costs, allowing for rapid prototyping of new ideas.

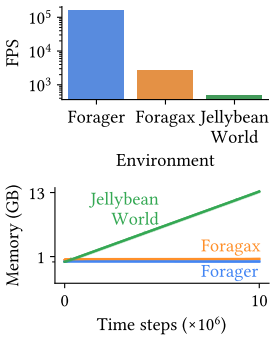


Figure 1: FPS and memory use of JBW and Forager.

AgarCL is a new benchmark based on the video game Agar.io that features multiple agents, changing dynamics, and a limited FOV (Mohamed et al., 2026). AgarCL remains an open challenging benchmark for continual reinforcement learning. In contrast to AgarCL's complex visual observations and hybrid action space, Forager deliberately uses simple grid world dynamics to achieve high simulation throughput to allow for rapid algorithmic development.

3 Background

In this paper, we consider continual reinforcement learning problems modeled as Markov Decision Processes (MDP)[1] where certain information is not visible to the agent. The agent's interaction proceeds synchronously, where on each discrete time-step $t = 1, 2, ...$ the agent chooses an action $A_t \in \mathcal{A}$ in state $S_t \in \mathcal{S}$. The environment then, in part due to the agent's action choice, transitions to a new state S_{t+1} and emits a scalar reward R_{t+1}. We assume that the environment is continuing, and performance is evaluated based on the average reward accumulated over the agent's lifetime.

We consider environments that use two approaches to limit agent observability. The first is to use non-stationary rewards, which is a form of partial observability because the dynamics producing the non-stationarity are hidden from the agent. A common approach to generate non-stationary rewards is to use a finite set of reward functions $\{R^{(0)}, R^{(1)}, ...\}$ and every τ time-steps simply switch to a different reward function. Typically τ, the switch events, and the set of possible rewards are not exposed to the agent. This setup can be framed as a multitask learning problem, but most researchers instead use such problems to measure how quickly agents can adjust to sudden, unexpected, unpredictable changes. Related, the reward can be changed with time ($R_{t+1} \doteq r_t(S_t, A_t, S_{t+1})$), typically slowly, producing an infinite sequence of problems. Another approach to limit agent observability is to limit the agent's field of view. There are other ways to make the MDP partially observable necessitating continual learning, such as changing the transition dynamics according to some unobservable function, which we explore in our last experiment.

We run demonstrative experiments with DQN (Mnih et al., 2015) and PPO (Schulman et al., 2017). We largely make use of function approximation architectures based on feedforward, fully connected neural networks but also investigate recurrent agents with DRQN (Hausknecht & Stone, 2015) and RTU-PPO (Elelimy et al., 2024).

4 The Forager environment

Forager is a grid-world environment where the agent collects a variety of mushrooms with different rewards and respawning behaviors. The environment simulates an infinite world from the agent's point of view: instead of being a simple grid with boundaries, it is a torus where the agent can go in the cardinal directions indefinitely by wrapping around the edges. The agent has a limited field of view (FOV), making the environment partially observable. Forager is actually a family of environments, configurable via user specified parameters. With a smaller FOV, the environment is more partially observable and challenging. At the other extreme, the FOV can be set to the size of the world, providing full observability. Further, the placement and respawning of objects, reward functions, and colors in the environment can be configured and changed over time.

Figure 2 shows an example Forager environment with two mushroom colors that can be collected by the agent. Figure 3 shows a different environment, where the agent forages for a variety of mushrooms, some high reward (dark brown), some lower reward (pink), and some poisonous and thus negative reward (yellow). The actions are {*up, down, left, right*} and move the agent in the corresponding cardinal direction. The blue square represents the agent, and the light blue overlay represents its FOV. When the agent collects an item, it disappears and the agent is rewarded. The objects either respawn in their original location or a random one according to user design. For example, in one of our experiments the high reward mushrooms respawn much more slowly than the other mushrooms, making them rare but more valuable. All other rewards are zero. Other collectible and unmovable objects (i.e., walls) can be easily added.

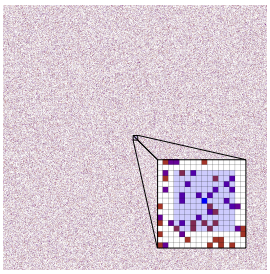


Figure 2: A sample foraging world.

[1] We do not introduce the more involved Partially Observable MDP notation, because it is not needed for this paper.

The observations are agent-centered, meaning that the agent is always at the center of the observation. The agent's observation is configurable to be either a RGB image of the agent's FOV or a binary tensor indicating the occupancy of each item type with shape (*FOV*, *FOV*, *unique objects*).

Non-stationary rewards are produced in multiple ways. One example is by decaying the reward for a mushroom on each step, simulating spoiling or rotting over time. Another is to have the value of a mushroom type change over time, simulating the agent's taste or nutritional needs changing over time. It might like one mushroom type in early life, and later in life, start preferring another. These non-stationary rewards would encourage agents to continually adapt to their changing environment.

5 Desiderata for a continual learning testbed

In this section we review the design principles of JBW (Platanios et al., 2020), the characteristics of good continual learning benchmarks Khetarpal et al. (2022), and the properties of continual learning problems motivated by a recent formal definition of continual learning (Abel et al., 2023).

The JBW environment predates most recent interest in CRL and was itself largely inspired by the work on the Never-ending Language Learning system (Carlson et al., 2010). The aforementioned system crawled the internet for years, reading and understanding webpages. The designers of JBW were inspired by the idea of unbounded learning systems, but also wanted to build a highly configurable environment where new experiments could be setup and run quickly. JBW is designed to require never-ending learning using two sources of partial observability: (1) a limited FOV in an infinite, procedurally generated world to navigate, and (2) non-stationary reward functions that either change slowly or periodically change completely. Forager was designed with the same principle, and differs in that it is more scalable than JBW and does not include some of advanced observation features like multi-modal observations (i.e., scent) and object occlusion.

Due to the lack of appropriate continual reinforcement learning benchmarks, others have proposed a set of properties that future benchmarks should have (Khetarpal et al., 2022). We mention them here as Forager does not achieve all these properties in full. In particular, Khetarpal et al. (2022) identify the following: 1) learning should be evaluated online and incrementally, 2) good performance should require discovery and composition of skills, 3) some form of physics which the agent can model, and 4) learnable causal dynamics such as object interactions. While our experiments with Forager do not demonstrate skill discovery and composition, it is not hard to imagine that skills for navigating to the edge of the FOV or the nearest mushroom would be useful for planning and acting. We only explore a few object types in this paper (consumable objects and walls), but more complex affordances like pushing or carrying objects could be easily added. Forager certainly does not implement interesting physics. We believe this is not worth the potential computational penalty and our experiments show Forager is a significant challenge for current algorithms.

Another good way to design an environment useful for continual learning is to look at how it matches formal notions of continual reinforcement learning. Abel et al. (2023) provides a formal problem specification that intuitively translates to problems where the best agents do not converge. In the stationary variants of Forager, even with a limited FOV, there are agents that can converge. However, it is a hard memory problem and for the limited agents we consider, it is likely necessary to track (i.e., learn forever). In the non-stationary variants of Forager, such as the one requiring never-ending learning (described below), the agent cannot model the partial observability and must track. Forager, therefore, is a continual learning problem according to this definition.

Forager largely achieves the desiderata laid out in the literature, just as JBW did before it. The key differences are motivated by the need for (1) a more lightweight environment and (2) to remove design features that are not critical to challenge current algorithms.

6 Investigating the impact of field of view

Forager provides an easy way to modify the degree of partial observability by altering the FOV observed by the agent. With a limited FOV, the agent must be able to use memory (or recurrent state updates) to remember the locations of rewarding mushrooms in the world beyond its FOV, and also predict when mushrooms will respawn outside of its FOV. In this section, we provide a simple demonstration of the effect of FOV.

The environment is visualized in Figure 3 (top). The idea is inspired by biomes, regions in the natural world where different flora and fauna live. After a foraging agent consumes all the valuable flora in one biome, it must navigate to the next one and continue foraging. This is made more challenging because the flora in different biomes do not regrow at the same rate. In this environment the agent is collecting mushrooms. The mushrooms on the left, morels, are high value (+30 reward), but respawn very slowly compared to the mushrooms on the right, which give rewards of +1 for oyster (pink) and -1 for deathcap (yellow) mushrooms. The mushrooms on the right respawn much more quickly. A smart agent should collect the morels whenever they are available and otherwise focus on collecting oyster mushrooms and avoiding deathcaps; continually switching back and forth.

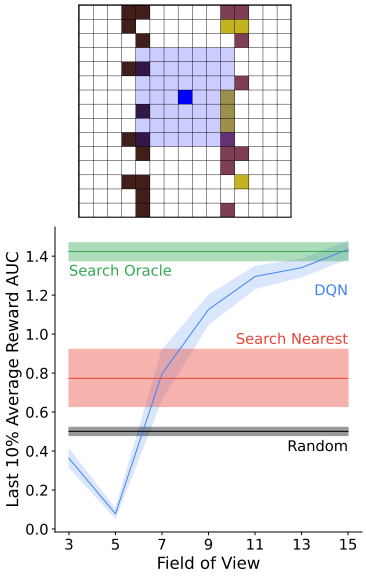


Figure 3: A simple Forager environment (ForagerTwoBiome-v1) with two biomes. The light blue overlay represents the FOV. The other colored squares are mushrooms which generate reward when collected. On the bottom we see how decreasing FOV makes the task more challenging for DQN. Results are averaged over 30 independent trials; shaded regions are 95% bootstrap CI.

In Figure 3 (bottom) we see that as the FOV size decreases, the environment becomes more difficult for DQN. With the largest FOV sizes, DQN performed similarly to the Search Oracle. From videos of agent behavior,[2] we see the expected behavior: DQN with larger FOV sizes acts almost exactly like the Search Oracle: foraging the oyster mushroom biome mostly, but periodically crossing the gap to forage the morels biome when they respawn. As the FOV size decreased, DQN performance deteriorated. For FOV size 7, DQN resembled Search Nearest in behavior, basically exclusively foraging oyster mushrooms and only rarely visiting the morel biome. The FOV sizes of 3 and 5 do not allow the agents to see across the gap. These agents performed worse than the baselines and stumbled around the world lacking reward-seeking behavior.

The results from this experiment were exactly as expected. With a large enough FOV, the environment is fully observable to the agent and DQN should and does learn near-optimal foraging. As we reduce the FOV, the agent cannot see when the morels have respawned, and therefore DQN would need to learn to take excursions into the left biome to check if the morels had respawned. We saw no evidence of this behavior emerging with smaller FOV sizes, which, again, is not surprising because this would require remembering the location of previously rewarding locations in the grid.

7 Investigating never-ending relearning in Forager

We expect a big world to require continual learning, exploration, and state construction from agents. The previous experiment does not actually require continual, unending learning, because everything is observable to the agent given enough interaction with the world—there is no hidden state-transition mechanism. In this section, we explore a switching-task variant of Forager that requires

[2] https://sites.google.com/view/rlc2026-forager/

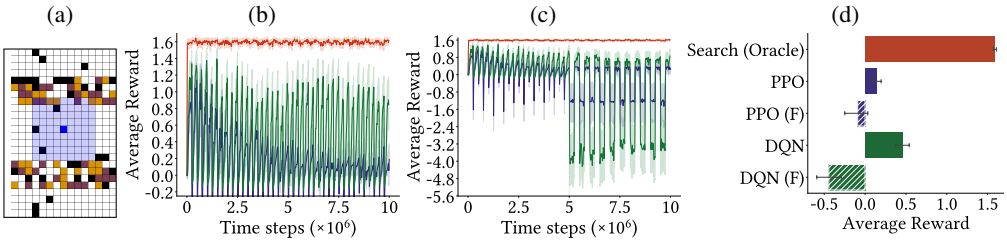


Figure 4: The performance, both learning curves and summary bar charts, of DQN and PPO. The bar chart serves as a legend for the learning curves. **(a)** Visualization of continual relearning task (ForagaxNeverEndingRelearning-v1). **(b)** We see both agents fall significantly short of the search baseline. DQN exhibits minor degradation over time, whereas PPO exhibits dramatic loss of plasticity. **(c)** However, by freezing learning halfway through the experiment, we see further drop in performance indicating both agents were indeed learning (poorly) to adapt to task switches. **(d)** The hatched bars report the performance of the agents frozen after 5 million steps, labeled (F).

unending policy change (for non-recurrent agents) to study the impact of recently proposed continual learning mitigation strategies. Then we investigate how state-construction methods support continual learning under constant task switching.

The setup of this experiment is similar to the previous 2-biome experiment, where there are two regions of interest with collectible mushrooms, but which mushrooms the agent should collect changes periodically and is region specific. The reward associated with each color is different on each side and this mapping changes periodically according to a hidden schedule. In the beginning, the agent should collect purple mushrooms (+4 reward) and avoid yellow (-2 reward) in the top biome and all mushrooms are negatively rewarding in the other biome (-8 for purple and -14 for yellow). After the switch, all the mushrooms generate negative rewards in the top biome (-14 for purple and -8 for yellow), but one of the mushroom types in the bottom biome generates positive reward and the agent should collect yellow (+4) and avoid purple (-2) there. Figure 4a provides a screen shot of the environment. This environment includes impassible and unmovable walls both to obstruct the agents and to help agents localize themselves. Since the mushrooms in both biomes are visually indistinguishable to the agent with a limited FOV (9), most agents are forced to constantly relearn their foraging policies after each switch. A recurrent agent could, in principle, learn to anticipate the switch and navigate between biomes more effectively. To help focus on investigating the continual learning aspect of this formulation, we one-hot encoded each color as the input to agents. In addition, all agents receive last reward and last action as part of their inputs.

The first question is can our non-recurrent deep RL methods continually learn in this setup? We tested DQN and PPO as representative value-based and policy gradient algorithms. We tuned the hyperparameters of all learning agents in this work using 10% tuning with 10 seeds, as tuning for the entire lifetime of the agent is not well aligned with the goals of CRL (Mesbahi et al., 2025). In Supplement 14.4.1, we outline all the hyperparameter ranges swept in this work. To contextualize the performance, we include a Search Oracle baseline that has full access to the underlying state (the precise location and reward for all objects in the world).

Figure 4b summarizes the results. All results are averaged over 30 independent runs and we report 95% bootstrap confidence intervals. We see DQN generally outperforms PPO, but does not reach the performance of the oracle, but as you will see later some learning methods perform much better than this. DQN appears to be suffering from minor loss of plasticity, while PPO is dramatically getting progressively worse with more task switches. Both DQN and PPO were indeed relying on continual learning. To see this clearly, we froze learning after 5 million steps in Figure 4c. We see a drop in performance of both agents; the frozen agents could only effectively collect mushrooms on whatever side they were frozen on.

8 Do mitigations help in Forager?

Next, we compared the impact of different mitigation strategies that have recently been proposed to promote CRL and prevent loss of plasticity. Since there are a large number of such mitigations, we investigate representatives from several categories, including: (1) regularizing the weights towards the network initialization (Kumar et al., 2024), (2) the CReLU activation that prevents neuron death (Abbas et al., 2023), (3) Shrink & Perturb (Ash & Adams, 2020), (4) L2 regularization (Dohare et al., 2024), (5) Permanent-Transient Q-learning (PT-DQN) (Anand & Precup, 2023) which uses multiple Q-functions to distill previous learning, and (6) ReDo, which resets dormant neurons (Sokar et al., 2023). All methods, including base DQN and PPO agents, include layer normalization, which appears to be beneficial for preventing plasticity loss (Lyle et al., 2024a). We restrict our focus to mitigations proposed for RL settings and exclude others that require task labels and task boundaries, like EWC (Kirkpatrick et al., 2017), as we are interested in approaches that can learn in the face of unexpected task changes.

Figure 5a shows the impact of these mitigations when combined with DQN and Figure 5b reports similar results with PPO. Note that 1) PT-DQN and ReDo were first demonstrated with DQN, so we did not extend them to PPO and 2) we did not test PPO with CReLU as PPO is almost exclusively used with Tanh. DQN's performance was significantly improved by CReLU, L2 Init, and ReDo, but not by PT-DQN, L2, nor Shrink & Perturb. It is worth noting that DQN did not exhibit a severe loss of plasticity and the mitigations mostly increased how quickly DQN adapted to task switches. For PPO, L2 Init completely mitigated loss of plasticity, L2 and Shrink & Perturb helped largely mitigate it. Across both algorithms, only L2 Init provided consistent benefit, though none of the mitigations provided sufficient benefit to get close to the search oracle.

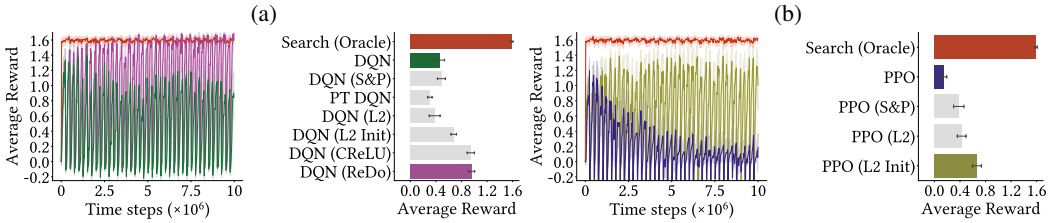


Figure 5: The impact of loss of plasticity mitigation strategies on DQN and PPO. **(a)** For DQN, only regularizing the weights to stay close to initialization (L2 Init), CReLU activations, and ReDo significantly outperform the base algorithm. **(b)** For PPO, Shrink & Perturb, L2, and L2 Init all help and the latter appears to prevent loss of plasticity. The learning curves only include the best mitigations to reduce visual clutter.

9 On the need for state-construction in CRL

The previous results are in some sense a mixed bag. PPO demonstrates loss of plasticity—the longer it learns, the worse it gets. DQN does not suffer in the same way. Both methods did demonstrate continual relearning required to outperform the naive search baselines. PPO performed significantly worse than the search baseline and, as you will see next, it is possible to perform better. Finally, while the mitigation strategies reduced loss of plasticity, they still underperformed the search baseline.

Forager, like any big world, requires both continual policy and value learning and state construction to do well. The switching structure certainly necessitates periodic relearning of the policy, but an agent with memory could do more. The limited FOV means there is significant aliasing in the agent's location, making navigating between biomes difficult. In addition, memory would help the agent anticipate the switches rather than relying on sampling negative rewarding mushrooms before adapting. While the mitigations in the previous section reduced loss of plasticity and slightly

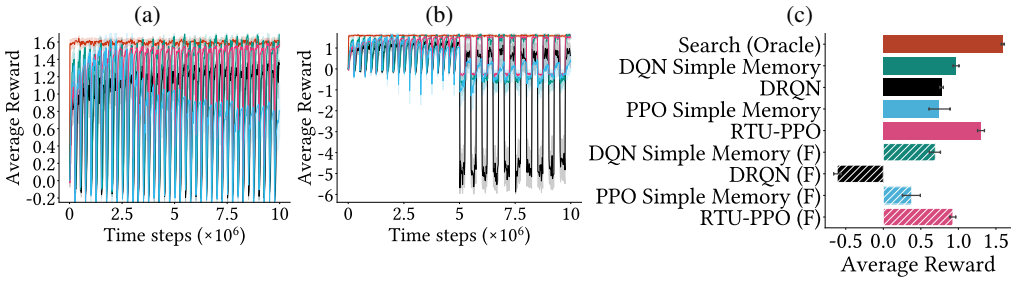


Figure 6: The performance of state construction methods in Forager. **(a)** We see RTU-PPO outperforms all other learning methods in this environment, DRQN outperforms DQN, and both DQN and PPO with a simple reward trace do well. **(b)** We see the impact of freezing these agents halfway through the experiment. All agents perform worse after freezing—indicating that they were indeed continually learning at the point of freezing. **(c)** Interestingly, the frozen performance of RTU-PPO and both simple memory approaches is better than all other methods indicating these methods have extracted some additional state information that makes the policy less dependent on tracking.

improved performance, we show that state construction not only mitigates loss of plasticity but also more effectively supports continual learning.

To highlight this we augmented the base agents with simple memory structures. We included an exponentially weighted memory trace (Rafiee et al., 2023) of the recent rewards as input observations to DQN and PPO. As highlighted in Figure 6a and 6b both agents exhibit significant performance improvements with the addition of simple memories of prior rewards. PPO with a simple memory still exhibits loss of plasticity but much more slowly than before. DQN with a simple memory at times matches the Search Oracle and equals the best mitigation, ReDo. In addition, comparing Figure 4d with Figure 6c, the frozen policy from DQN with simple memory outperforms the frozen policy from vanilla DQN.

Exponential memories are extremely limited, so we also investigated the impact of recurrent variants of DQN and PPO. We tested a modified variant of the DRQN algorithm (Hausknecht & Stone, 2015) which combines DQN with an RNN trained via truncated backprop through time (TBPTT). We modernized DRQN to use gated recurrent units (GRU) and stored the hidden state in the buffer and employed a hidden state warm up to mitigate the impact of stale states in TBPTT similar to Kapturowski et al. (2019). We also include a recent recurrent PPO algorithm with a RTU layer that uses a diagonal approximation and a sin-cosine encoding to allow fast realtime recurrent learning updates (Elelimy et al., 2024).

The results are fairly striking. DRQN improves the performance of DQN but still falls short of the simple memory approach and RTU dramatically improves the performance of PPO. Loss of plasticity is eliminated and this agent achieves the best performance of all learning agents—better than DQN with ReDo—very close to the search baseline with low variance in performance. This exceptional performance is not because RTU-PPO has extracted the underlying state and converged to a static high performance policy. As we see in Figure 6b: RTU-PPO when frozen exhibits a significant performance drop, thus providing clear evidence that this agent is indeed continually relearning its policy of every switch, in addition to achieving better performance due to the recurrent network mitigating partial observability.

10 An unending sequence of foraging tasks

In this section, we introduce a challenge problem for CRL: an environment with an unending sequence of foraging tasks which change based on the agent's actions and unobservable timeseries. As visualized in Figure 7a, there are four biomes, each with its own mushroom species. Each mush-

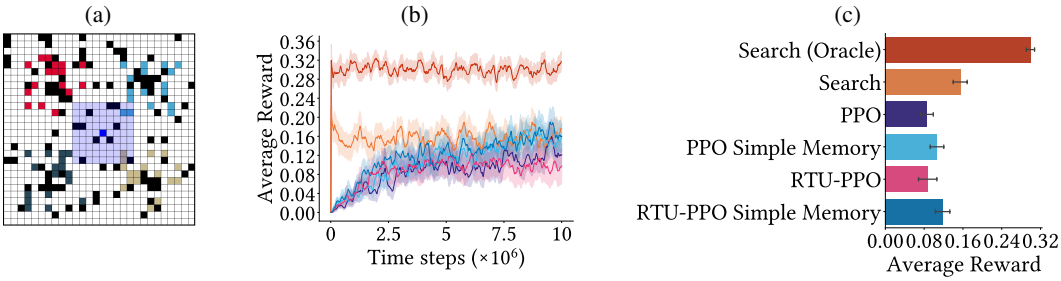


Figure 7: Foraging experiment with an unending series of tasks (ForagaxUnendingTasks-v1). **(a)** Environment visualization: the blue square is the agent; the red, light blue, dark gray, and beige squares are different mushroom species; and the black squares are walls. **(b)** The learning curves of several learning agents. **(c)** The average reward obtained by each method. RTU-PPO Simple Memory combines both RTU and simple memory approach. The Search agent performs local search within the same FOV as other agents. The Search Oracle agent has access to the entire grid. Both search agents have privileged information about the current reward. All results are averaged over 30 independent trials and the shaded regions and error bars represent 95% bootstrap CI.

room species has a randomly sampled color and reward function that fluctuates with time based on a unique time series described in Supplement 14.5. Every 100 steps in the environment, a global cue signals the biome with the best rewarding mushrooms for 10 steps. The task switches are driven by agent behavior. When an agent consumes 10,000 individuals of a mushroom species, that species goes extinct and is replaced by a new species with a new color and new reward, forcing the agent to continually adapt its foraging strategy while also remembering the cue.

We focus our evaluation on PPO-based agents, as the previous experiments demonstrated that RTU-PPO outperformed DQN-based agents and other mitigations. We use a convolutional neural network to handle RGB input. Additional experiment details are available in Supplement 14.5.

Figure 7 showcases the results of this experiment. While the learning agents do not exhibit loss of plasticity, their performance does not improve beyond the search nearest agent within 10 million steps of training. All learning agents were unable to reach the performance level of the Search Oracle baseline. The results demonstrate that RTU-PPO was unable to mitigate the partial observability and construct an agent state that supports rapid adaptation to the unending stream of tasks more effectively than Simple Memory. These results are not surprising because this task is much harder than the 2-biome variants. The increased rate of reward non-stationarity requires the agent to adapt quickly between tasks while also constructing state. The larger world size and limited FOV also requires the agent to explore and perform state construction to enable localization while remembering the cue. Contrasting our results with those of Simmons-Edler et al. (2025) in Craftax, we see the additional challenge posed by Forager's synchronous learning (i.e., a single stream of experience)—our CRL algorithms are still fairly data inefficient.

This challenging variant of Forager opens many avenues for algorithmic development. The presence of partial observability and rapidly fluctuating rewards poses a unique challenge for continual learning with recurrent RL algorithms. The environment has a biome structure that makes it interesting for investigating temporal abstraction, option discovery, and model learning. Lastly, the larger world size highlights the need for effective continual exploration methods that enable agents to quickly navigate and discover the best biome. We could also make the environment more interesting and possibly more challenging by replacing the time-series behind each reward signal with timeseries from the real world, like the temperature in different cities. This computationally inexpensive environment offers a unique challenge problem for driving further research progress in CRL.

11 Conclusion

In this paper we introduced a new continual learning testbed called Forager, that is ultra-lightweight and allows for rapid prototyping. This testbed is a family of environments, with configurable parameters on the size of the environment, level of partial observability and configurable objects with optionally non-stationary rewards. We proposed a variety of different tasks, provided sensible baselines for each that makes performance interpretable and tested several RL algorithms in these environments. We showed how algorithms shift from performing well to failing as the FOV is changed. We demonstrated that Forager can be used to study loss of plasticity and algorithms to mitigate it. We also showed how state construction via recurrent architectures play an important role in CRL. Finally, we showed that a surprising-level of complexity can be introduced into this relatively bounded environment, simply through structured spawning and incorporating non-stationary observations and rewards. This environment is one where we can design agents to solve particular tasks, without the risk of overfitting, because new tasks are easy to create and likely to pose new challenges. Our experiments in Forager have highlighted that our existing algorithms have clear limitations, and that Forager can be both a challenge problem and a scientific testbed.

Broader Impact Statement

This paper proposes a testbed for CRL research, that is neither realistic, nor of commercial interest. This work very much targets fundamental advances in RL algorithms and empirical practices, thus there are no concerns with broader impact.

References

Zaheer Abbas, Rosie Zhao, Joseph Modayil, Adam White, and Marlos C Machado. Loss of plasticity in continual deep reinforcement learning. In *Conference on Lifelong Learning Agents*, 2023.

David Abel, André Barreto, Benjamin Van Roy, Doina Precup, Hado P van Hasselt, and Satinder Singh. A definition of continual reinforcement learning. *Advances in Neural Information Processing Systems*, 2023.

Ossama Ahmed, Frederik Träuble, Anirudh Goyal, Alexander Neitz, Manuel Wüthrich, Yoshua Bengio, Bernhard Schölkopf, and Stefan Bauer. Causalworld: A robotic manipulation benchmark for causal structure and transfer learning. In *International Conference on Learning Representations*, 2021.

Nishanth Anand and Doina Precup. Prediction and control in continual reinforcement learning. *Advances in Neural Information Processing Systems*, 2023.

Jordan Ash and Ryan P Adams. On warm-starting neural network training. *Advances in Neural Information Processing Systems*, 2020.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 2013.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam Hruschka, and Tom Mitchell. Toward an architecture for never-ending language learning. In *AAAI Conference on Artificial Intelligence*, 2010.

Shibhansh Dohare, J. Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A. Rupam Mahmood, and Richard S Sutton. Loss of plasticity in deep continual learning. *Nature*, 2024.

Esraa Elelimy, Adam White, Michael Bowling, and Martha White. Real-time recurrent learning using trace units in reinforcement learning. *Advances in Neural Information Processing Systems*, 2024.

Mohamed Elsayed and A. Rupam Mahmood. Addressing loss of plasticity and catastrophic forgetting in continual learning. In *International Conference on Learning Representations*, 2024.

Alexandre Galashov, Michalis Titsias, András György, Clare Lyle, Razvan Pascanu, Yee Whye Teh, and Maneesh Sahani. Non-stationary learning of neural networks with automatic soft parameter reset. *Advances in Neural Information Processing Systems*, 2024.

William H. Guss, Mario Ynocente Castro, Sam Devlin, Brandon Houghton, Noboru Sean Kuno, Crissman Loomis, Stephanie Milani, Sharada P. Mohanty, Keisuke Nakata, Ruslan Salakhutdinov, John Schulman, Shinya Shiroshita, Nicholay Topin, Avinash Ummadisingu, and Oriol Vinyals. The minerl 2020 competition on sample efficient reinforcement learning using human priors. *arXiv preprint arXiv:2101.11071*, 2021.

Danijar Hafner. Benchmarking the spectrum of agent capabilities. In *International Conference on Learning Representations*, 2022.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 2025.

Matthew J Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *AAAI Fall Symposia*, 2015.

Muhammad Kamran Janjua, Haseeb Shah, Martha White, Erfan Miahi, Marlos C Machado, and Adam White. Gvfs in the real world: making predictions online for water treatment. *Machine Learning*, 2023.

Khurram Javed and Richard S Sutton. The big world hypothesis and its ramifications for artificial intelligence. In *Finding the Frame: An RLC Workshop for Examining Conceptual Frameworks*, 2024.

Steven Kapturowski, Georg Ostrovski, Will Dabney, John Quan, and Remi Munos. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2019.

Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 2022.

James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 2017.

Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *International Conference on Learning Representations*, 2020.

Saurabh Kumar, Henrik Marklund, and Benjamin Van Roy. Maintaining plasticity in continual learning via regenerative regularization. In *Conference on Lifelong Learning Agents*, 2024.

Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. *Advances in Neural Information Processing Systems*, 2020.

Hojoon Lee, Hanseul Cho, Hyunseung Kim, Daehoon Gwak, Joonkee Kim, Jaegul Choo, Se-Young Yun, and Chulhee Yun. Plastic: Improving input and label plasticity for sample efficient reinforcement learning. *Advances in Neural Information Processing Systems*, 2023.

Hojoon Lee, Hyeonseo Cho, Hyunseung Kim, Donghu Kim, Dugki Min, Jaegul Choo, and Clare Lyle. Slow and steady wins the race: maintaining plasticity with hare and tortoise networks. In *International Conference on Learning Representations*, 2024.

Alex Lewandowski, Michał Bortkiewicz, Saurabh Kumar, András György, Dale Schuurmans, Mateusz Ostaszewski, and Marlos C Machado. Learning continually by spectral regularization. In *International Conference on Learning Representations*, 2025.

Clare Lyle, Mark Rowland, and Will Dabney. Understanding and preventing capacity loss in reinforcement learning. In *International Conference on Learning Representations*, 2022.

Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks. In *International Conference on Machine Learning*, 2023.

Clare Lyle, Zeyu Zheng, Khimya Khetarpal, James Martens, Hado van Hasselt, Razvan Pascanu, and Will Dabney. Normalization and effective learning rates in reinforcement learning. In *Advances in Neural Information Processing Systems*, 2024a.

Clare Lyle, Zeyu Zheng, Khimya Khetarpal, Hado van Hasselt, Razvan Pascanu, James Martens, and Will Dabney. Disentangling the causes of plasticity loss in neural networks. In *Conference on Lifelong Learning Agents*, 2024b.

Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Thomas Jackson, Samuel Coward, and Jakob Nicolaus Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning. In *International Conference on Machine Learning*, 2024.

Walter Mayor, Johan Obando-Ceron, Aaron Courville, and Pablo Samuel Castro. The impact of on-policy parallelized data collection on deep reinforcement learning networks. In *International Conference on Machine Learning*, 2025.

Golnaz Mesbahi, Parham Mohammad Panahi, Olya Mastikhina, Steven Tang, Martha White, and Adam White. Position: Lifetime tuning is incompatible with continual reinforcement learning. In *International Conference on Machine Learning Position Paper Track*, 2025.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

Mohamed A. Mohamed, Kateryna Nekhomiazh, Vedant Vyas, Marcos M. Jose, Andrew Patterson, and Marlos C. Machado. The cell must go on: Agar.io for continual reinforcement learning. *arXiv preprint arXiv:2505.18347*, 2026.

Evgenii Nikishin, Junhyuk Oh, Georg Ostrovski, Clare Lyle, Razvan Pascanu, Will Dabney, and André Barreto. Deep reinforcement learning with plasticity injection. *Advances in Neural Information Processing Systems*, 2023.

Andrew Patterson, Samuel Neumann, Martha White, and Adam White. Empirical design in reinforcement learning. *Journal of Machine Learning Research*, 2024.

Emmanouil Antonios Platanios, Abulhair Saparov, and Tom Mitchell. Jelly bean world: A testbed for never-ending learning. In *International Conference on Learning Representations*, 2020.

Sam Powers, Eliot Xing, Eric Kolve, Roozbeh Mottaghi, and Abhinav Gupta. Cora: Benchmarks, baselines, and metrics as a platform for continual reinforcement learning agents. In *Conference on Lifelong Learning Agents*, 2022.

Banafsheh Rafiee, Zaheer Abbas, Sina Ghiassian, Raksha Kumaraswamy, Richard S Sutton, Elliot A Ludvig, and Adam White. From eye-blinks to state construction: Diagnostic benchmarks for online representation learning. *Adaptive Behavior*, 2023.

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

Archit Sharma, Kelvin Xu, Nikhil Sardana, Abhishek Gupta, Karol Hausman, Sergey Levine, and Chelsea Finn. Autonomous reinforcement learning: Formalism and benchmarking. In *International Conference on Learning Representations*, 2022.

Riley Simmons-Edler, Ryan P Badman, Felix Baastad Berg, Raymond Chua, John J Vastola, Joshua Lunger, William Qian, and Kanaka Rajan. Deep rl needs deep behavior analysis: Exploring implicit planning by model-free agents in open-ended environments. *arXiv preprint arXiv:2506.06981*, 2025.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *International Conference on Machine Learning*, 2023.

Richard S Sutton, Anna Koop, and David Silver. On the role of tracking in stationary environments. In *International Conference on Machine Learning*, 2007.

Chen Tessler, Shahar Givony, Tom Zahavy, Daniel Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI Conference on Artificial Intelligence*, 2017.

Tristan Tomilin, Meng Fang, Yudi Zhang, and Mykola Pechenizkiy. Coom: A game benchmark for continual reinforcement learning. *Advances in Neural Information Processing Systems*, 2023.

Supplementary Materials

The following content was not necessarily subject to peer review.

12 Benchmarking compute and memory in Jelly Bean World and Forager

We benchmark Forager CPU implementation, Forager JAX implementation (Foragax), and Jelly Bean World in the large-scale foraging setup described in Section 13 with a constant policy that always uses the up action for 10 million time steps. The metrics are averaged over 5 independent sequential runs on a desktop machine.

There are many factors such as world size, the complexity of environment dynamics, and hardware that impact benchmarking results. Consequently, these figures should be viewed as a relative performance comparison within the context of the large-scale foraging setup.

Environment	Wall time (h)	Speed (FPS)	Memory (GB)
Forager	0.02	159879	0.1
Foragax	1.04	2678	0.6
Jelly Bean World	5.70	487	13.2

Table 1: Resource utilization comparison across Forager, Foragax, and Jelly Bean World.

The benchmarking results in Table 1 demonstrate that for this large-scale foraging setup, Foragax was 5 times faster than JBW and Forager was 328 times faster than JBW. In Figure 1, we note that Forager and Foragax exhibited constant memory usage, while Jelly Bean World's memory usage increased linearly with the number of time steps.

13 JBW-like experiments in Forager

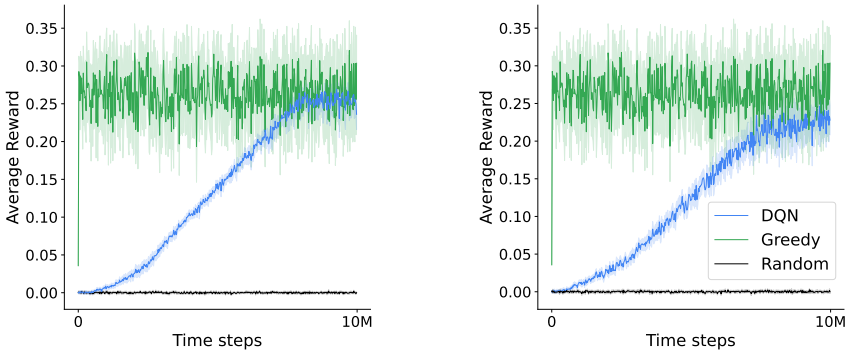
JBW introduced the idea of running experiments in an unbounded world to explore never-ending learning problems. However, it is the unbounded nature of JBW that also causes issues with unbounded memory usage as the agent explores the world. A natural question to ask is whether large scale foraging experiments typically done in JBW produce similar learning dynamics in Forager with a fixed world size. To address this question, we set up a nearly-identical experiment in JBW and Forager to demonstrate that JBW-like large-scale foraging experiments can also be run in Forager.

First, let us describe the setup of the experiment in Forager. We take inspiration from the non-episodic case study introduced by the original work (Platanios et al., 2020) and set up an experiment with rewarding objects (jelly beans) and negative reward objects (onions), with rewards of +1 and -1 respectively. The objects spawn uniformly with 0.1 probability. The Forager world size is 1000×1000 with wrapping and mushrooms respawn in random locations to approximate the infinite procedurally generated world in JBW. This environment, which we denote as Forager Extra Large, is visualized in Figure 2. The discount factor is 0.99. We evaluate agents in our experiments using the exponential moving average of reward, with a decay of 0.999. This measure is similar to the reward rate used in previous work (Platanios et al., 2020) and measures an agent's ability to adapt to the current circumstances. The FOV is set to 11, resulting in $(11 \times 11 \times 2)$ observations with each channel encoding the absence or presence of an object.

In both JBW and Forager we ran the same set of agents. We ran DQN (Mnih et al., 2015) with a convolutional layer with 16 kernels with size 3×3 with stride 1, followed by a hidden layer with width 64 and ReLU activation, then a final linear layer of size 4. We sweep the step size of each agent from $\{10^{-3}, 3 \times 10^{-4}, 10^{-4}, 3 \times 10^{-5}, 10^{-5}\}$ for 100 K steps averaging the total reward (area under the curve, AUC) over five runs (so called k% tuning (Mesbahi et al., 2025)). The hyperparameters are detailed in Tables 2 and 3. We then evaluate each agent for 10 M steps with 30 independent trials with 30 random seeds, different from those used in the hyperparameter sweep (a two stage

procedure (Patterson et al., 2024)). We also ran two baselines: one that selects actions uniformly (Random) and one that uses breadth first search to navigate greedily towards the closest jelly bean while avoiding onions (Greedy).

Figure 8 shows the learning curves of DQN and the two baselines were fairly similar between JBW and Forager. DQN eventually performed at a similar level to the search baseline in both environments. The environment and challenge posed by Forager appears to be quite similar to JBW. This is a large Forager environment when you consider the size of the FOV. We also demonstrate that simple DQN agents are able to learn effective foraging for 10 M steps without underperforming later (no obvious loss of plasticity). The lightweight design of Forager allows us to run longer experiments and more independent trials with the same amount of compute. Overall, for large-scale foraging environments, Forager can serve as a more efficient drop-in replacement for JBW.



(a) Jelly Bean World

(b) Forager Extra Large

Figure 8: Both JBW and Forager can be used to conduct large-scale gridworld foraging experiments. Results are averaged over 30 independent trials and the shaded regions are 95% bootstrap CI.

14 Experiment Details

14.1 Large scale foraging experiment

Table 2: Hyperparameter choices and defaults for DQN in large scale foraging experiment

Hyperparameter	**Choices**
Step size	$\{10^{-3}, 3 \times 10^{-4}, 10^{-4}, 3 \times 10^{-5}, 10^{-5}\}$
Hyperparameter	**Default**
Exploration Strategy	ϵ-greedy with linear decay
ϵ-greedy initial ϵ	1.0
ϵ-greedy final ϵ	0.05
ϵ-greedy decay percentage	80%
Update frequency	4
Minibatch size	32
Replay memory size	10000
Minimum replay history	32
Buffer sampling strategy	uniform
Target network update frequency	128
Discount factor γ	0.99
Optimizer	Adam
Adam β_1	0.9
Adam β_2	0.999
Adam ϵ	10^{-8}

Table 3: Selected hyperparameters for DQN in large scale foraging experiment

Environment **Hyperparameter**	**Forager Extra Large**	**Jelly Bean World**
Step size	10^{-3}	10^{-3}

14.2 Field of view experiment

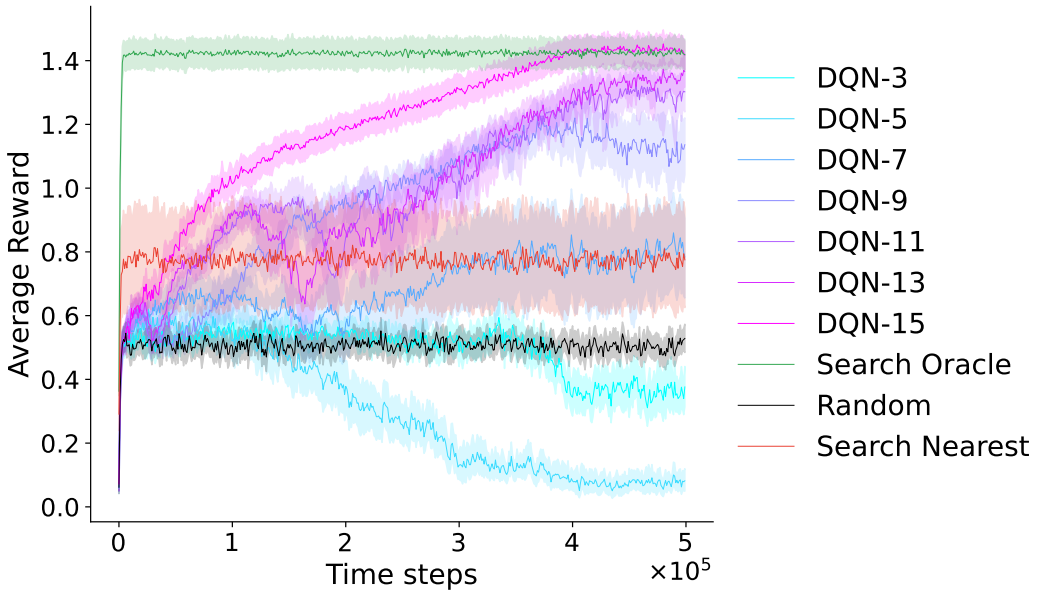


Figure 9: Morel 2-biome

Figure 10: The effect of FOV on average reward learning curves of DQN agents. Agents with larger FOV sizes outperform those with smaller FOVs. Results are averaged over 30 independent trials and the shaded regions are 95% bootstrap CI.

Table 4: Hyperparameter choices and defaults for DQN in FOV experiments

Hyperparameter	**Choices**
Step size	$\{10^{-3}, 3 \times 10^{-4}, 10^{-4}, 3 \times 10^{-5}, 10^{-5}\}$
Update frequency	$\{1, 4\}$
Target network update frequency	$\{1, 128\}$
Adam β_2	$\{0.9, 0.999\}$
Adam ϵ	$\{0.01, 10^{-8}\}$
Hyperparameter	**Default**
Exploration Strategy	ϵ-greedy with linear decay
ϵ-greedy initial ϵ	1.0
ϵ-greedy final ϵ	0.05
ϵ-greedy decay percentage	80%
Minibatch size	32
Replay buffer size	10000
Minimum replay history	32
Buffer sampling strategy	uniform
Discount factor γ	0.99
Optimizer	Adam
Adam β_1	0.9

Table 5: Selected hyperparameters for DQN in FOV experiment

FOV / **Hyperparameter**	**3**	**5**	**7**	**9**	**11**	**13**	**15**
Step size	10^{-3}	10^{-3}	10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}
Update frequency	4	4	1	4	4	4	4
Target network update frequency	1	1	1	128	128	128	128
Adam β_2	0.999	0.9	0.999	0.9	0.9	0.9	0.9
Adam ϵ	0.01	0.01	0.01	10^{-8}	10^{-8}	10^{-8}	10^{-8}

14.3 Never-ending relearning experiments

For DQN, the input observation is flattened and concatenated with the previous action and reward to form the input vector. If reward trace is enabled, it is also concatenated.

The DQN neural network architecture consists of a MLP with two hidden layers of 64 hidden units, each with LayerNorm and ReLU activation, followed by a linear projection to output 4 action-values.

The DRQN neural network adds a GRU with 64 hidden units between the two hidden layers. The input to the GRU is concatenated to its output to improve gradient flow.

The PPO neural network architecture consists of independent actor and critic networks with the same neural network body architecture. The neural network body has a single layer MLP encoder with 64 hidden units, LayerNorm, and Tanh activation which outputs latent features that are then concatenated with the previous action and reward. If reward trace is enabled, it is also concatenated. The concatenated embedding is followed by a hidden layer with 512 hidden units, and then a hidden layer with 64 hidden units with LayerNorm and Tanh activation. The actor has a linear projection head to output 4 action preferences. The critic has a linear projection head to output a value.

The RTU-PPO neural network follows the structure of the PPO neural network with one difference. The 512 unit hidden layer is replaced with Recurrent Trace Units (RTU) (Elelimy et al., 2024) with 512 trace units, followed by ReLU. The input to the RTU is concatenated to its output to improve gradient flow. It is to be noted that RTU outputs two coupled values per trace unit, so the output directly from RTU is 1024.

14.4 State-construction experiments

14.4.1 Hyperparameters

Tables 6 and 7 contain the hyperparameters used for DQN variants; likewise, Tables 8 and 9 contain the hyperparameters used for PPO variants.

The PT-DQN algorithm Anand & Precup (2023) uses 64 hidden units for both the permanent and transient networks.

Table 6: Swept and agent-specific hyperparameters for DQN-based agents in never-ending relearning experiments. All agents sweep step size $\alpha \in \{3 \times 10^{-3}, 10^{-3}, 3 \times 10^{-4}, 10^{-4}, 3 \times 10^{-5}\}$ and $\epsilon \in \{0.05, 0.1, 0.25\}$ unless noted.[†]

Agent	Hyperparameter	Choices / Selected
DQN	α	3×10^{-3}
	ϵ	0.1
DQN (CReLU)	α	3×10^{-3}
	ϵ	0.1
DQN (L2)	α	3×10^{-3}
	ϵ	0.1
	$\lambda_{\text{L2}} \in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$	10^{-5}
DQN (L2 Init)	α	10^{-3}
	ϵ	0.1
	$\lambda_{\text{L2 Init}} \in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$	10^{-5}
DQN (ReDo)	α	3×10^{-3}
	ϵ	0.1
	$\tau_{\text{ReDo}} \in \{0, 0.01, 0.025, 0.1\}$	0.01
	ReDo frequency	10,000
DQN (S&P)	α	10^{-3}
	ϵ	0.25
	Shrink factor $\in \{0.8, 0.9\}$	0.9
	Noise scale $\in \{0.01, 0.001\}$	0.01
	S&P frequency	10,000
DQN (Simple Memory)	α	3×10^{-3}
	ϵ	0.1
	Decay $\in \{0.9, 0.99\}$	0.9
DRQN	α	10^{-4}
	ϵ	0.25
	TBPTT length	16
	Burn-in steps	16
PT-DQN	α	10^{-3}
	ϵ	0.1
	PT decay $\in \{0.55, 0.75, 0.95\}$	0.95
	PT α ratio $\in \{10^{-3}, 10^{-4}, 10^{-5}\}$	10^{-5}
	PT frequency	10,000
	PT replay memory size	10,000
	PT optimizer	SGD

Table 7: Shared default hyperparameters for all DQN-based agents in never-ending relearning experiments.

Hyperparameter	**Value**
Exploration strategy	ϵ-greedy
Update frequency	4
Minibatch size	32
Replay memory size	1,000
Minimum replay history	50
Buffer sampling strategy	uniform
Target network update frequency	128
Discount factor γ	0.99
Optimizer	Adam
Adam β_1	0.9
Adam β_2	0.999
Adam ϵ	10^{-5}

Table 8: Swept and agent-specific hyperparameters for PPO-based agents in never-ending relearning experiments. All agents sweep actor step size $\alpha_a \in \{10^{-3}, 3 \times 10^{-4}, 10^{-4}\}$, Critic step size scale factor $\in \{0.1, 1.0, 10.0\}$ (the critic learning rate α_c is set to $\alpha_a \times$ Critic step size scale factor), and entropy coefficient $\in \{0.01, 0.1, 1.0\}$.

Agent	**Hyperparameter**	**Choices / Selected**
PPO	α_a	3×10^{-4}
	Critic step size scale factor	0.1
	Entropy coefficient	0.01
PPO (L2)	α_a	10^{-3}
	Critic step size scale factor	0.1
	Entropy coefficient	0.01
	$\lambda_{\text{L2}} \in \{10^{-2}, 10^{-3}, 10^{-4}\}$	10^{-4}
PPO (L2 Init)	α_a	10^{-3}
	Critic step size scale factor	0.1
	Entropy coefficient	0.01
	$\lambda_{\text{L2 Init}} \in \{10^{-2}, 10^{-3}, 10^{-4}\}$	10^{-4}
PPO (S&P)	α_a	10^{-3}
	Critic step size scale factor	1.0
	Entropy coefficient	0.01
	Shrink factor $\in \{0.8, 0.9\}$	0.9
	Noise scale $\in \{0.01, 0.001\}$	0.01
	S&P interval	10,000 steps
PPO (Simple Memory)	α_a	10^{-3}
	Critic step size scale factor	0.1
	Entropy coefficient	0.01
	Decay $\in \{0.9, 0.99\}$	0.9
RTU-PPO	α_a	3×10^{-4}
	Critic step size scale factor	10.0
	Entropy coefficient	0.1

Table 9: Shared default hyperparameters for all PPO-based agents in never-ending relearning experiments.

Hyperparameter	**Value**
Rollout steps	2,048
Epochs	4
Number of mini-batches	32
Clipping ϵ	0.2
Max gradient norm	0.5
Value function coefficient	0.5
GAE λ	0.95
Discount factor γ	0.99
Optimizer	Adam
Adam β_1	0.9
Adam β_2	0.999
Adam ϵ	10^{-5}

14.5 Unending tasks experiment

The reward function for each biome used is $r(t) = \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi n}{T} \left\lfloor \frac{t}{w} \right\rfloor\right) + b_n \sin\left(\frac{2\pi n}{T} \left\lfloor \frac{t}{w} \right\rfloor\right)\right)$, with parameters $N = 10$, reward repeat $w = 1000$ (the reward is held constant for 1000 environment steps). Each biome independently samples the parameters $a_n, b_n \sim \mathcal{N}\left(0, \frac{1}{n^2}\right)$, the period T is sampled uniformly from $\{10, 11, \ldots, 1000\}$ (the reward function cycles after Tw environment steps). Each biome's raw Fourier sum is then min-max normalized to $[-1, 1]$ using the extrema over one full cycle, so every mushroom species has the same reward range regardless of the sampled a_n, b_n; the sampled parameters controls the shape of the reward curve rather than its amplitude. All the reward time series in the environment are centered at each time step by subtracting the mean reward over all mushrooms to ensure that there is always at least one mushroom species with non-negative rewards. Mushrooms respawn randomly within a biome with respawn time m sampled uniformly from $\{9, 10\}$ steps. Walls are scattered through the environment to aid an agent to locate itself within the large world and force it to learn how to navigate around them.

We use 10 independent trials with 10%-tuning to identify the best hyperparameters. We evaluate each agent for 10 M steps with 30 independent trials.

The PPO and RTU-PPO neural network architectures are modified by the addition of a convolutional vision encoder to process RGB images. The convolutional vision encoder consists of a pointwise convolutional layer (16 filters with kernel size 1×1 with a stride of 1, followed by LayerNorm and respective activation), a convolutional layer (16 filters with kernel size 3×3 with a stride of 1, followed by LayerNorm and respective activation). In preliminary experiments, a pointwise convolutional layer provided useful inductive bias for learning. The resulting vision features are flattened into a 1D vector. Following the vision encoder is a linear layer with 64 hidden units, LayerNorm, and Tanh activation. Then the global auditory cue vector is concatenated along with the previous action and reward to the output of the encoder. For PPO, this is followed by a linear layer with 1024 hidden units with LayerNorm and Tanh activation. For RTU-PPO, this is followed by a layer of Recurrent Trace Units (RTU) (Elelimy et al., 2024) with 1024 trace units, followed by ReLU activations. The input to the RTU is concatenated to its output to improve gradient flow. In both network architectures, this is followed by another linear layer with 64 hidden units with LayerNorm and Tanh activation. The actor has a linear projection head to output 4 action preferences. The critic has a linear projection head to output a value.

Table 10: Hyperparameter choices and defaults for PPO in Unending Tasks Forager experiment

Hyperparameter	**Choices**	**Selected**	**Selected (Simple Memory)**
Actor step size	$\{10^{-3}, 3 \times 10^{-4}, 10^{-4}\}$	3×10^{-4}	10^{-4}
Entropy coefficient	$\{0.01, 0.1, 1.0\}$	0.1	0.1
Critic step size scale factor	$\{0.1, 1.0, 10\}$	1	1
Decay	$\{0.9, 0.99\}$	N/A	0.9

Hyperparameter	**Default**
Rollout steps	128
Epochs per update	4
Number of minibatches	1
Clipping ϵ	0.2
Max gradient norm	0.5
Value function coefficient	0.5
GAE λ	0.95
Discount factor γ	0.99
Optimizer	Adam
Adam β_1	0.9
Adam β_2	0.999
Adam ϵ	10^{-5}

Table 11: Hyperparameter choices and defaults for RTU-PPO in Unending Tasks Forager experiment

Hyperparameter	**Choices**	**Selected**	**Selected (Simple Memory)**
Actor step size	$\{10^{-3}, 3 \times 10^{-4}, 10^{-4}\}$	10^{-4}	10^{-4}
Entropy coefficient	$\{0.01, 0.1, 1.0\}$	0.1	0.1
Critic step size scale factor	$\{0.1, 1.0, 10\}$	10	1
Decay	$\{0.9, 0.99\}$	N/A	0.99

Hyperparameter	**Default**
Rollout steps	128
Epochs per update	4
Number of minibatches	1
Clipping ϵ	0.2
Max gradient norm	0.5
Value function coefficient	0.5
GAE λ	0.95
Discount factor γ	0.99
Optimizer	Adam
Adam β_1	0.9
Adam β_2	0.999
Adam ϵ	10^{-5}