

# CORA: BENCHMARKS, BASELINES, AND METRICS AS A PLATFORM FOR CONTINUAL REINFORCEMENT LEARNING AGENTS

**Sam Powers\***

Carnegie Mellon University  
snpowers@cs.cmu.edu

**Eliot Xing\***

Georgia Institute of Technology  
exing@gatech.edu

**Eric Kolve**

Allen Institute for AI

**Roozbeh Mottaghi**

Allen Institute for AI

**Abhinav Gupta**

Carnegie Mellon University

## ABSTRACT

Progress in continual reinforcement learning has been limited due to several barriers to entry: missing code, high compute requirements, and a lack of suitable benchmarks. In this work, we present CORA, a platform for Continual Reinforcement Learning Agents that provides benchmarks, baselines, and metrics in a single code package. The benchmarks we provide are designed to evaluate different aspects of the continual RL challenge, such as catastrophic forgetting, plasticity, ability to generalize, and sample-efficient learning. Three of the benchmarks utilize video game environments (Atari, Procgen, NetHack). The fourth benchmark, CHORES, consists of four different task sequences in a visually realistic home simulator, drawn from a diverse set of task and scene parameters. To compare continual RL methods on these benchmarks, we prepare three metrics in CORA: Continual Evaluation, Isolated Forgetting, and Zero-Shot Forward Transfer. Finally, CORA includes a set of performant, open-source baselines of existing algorithms for researchers to use and expand on. We release CORA and hope that the continual RL community can benefit from our contributions, to accelerate the development of new continual RL algorithms.

## 1 INTRODUCTION

Over the course of the last decade, reinforcement learning (RL) has developed into a promising tool for learning a large variety of tasks, such as robotic manipulation (Kober & Peters, 2012; Kormushev et al., 2013; Deisenroth et al., 2013; Parisi et al., 2015), embodied AI (Zhu et al., 2017; Shridhar et al., 2020; Batra et al., 2020; Szot et al., 2021), video games (Mnih et al., 2015; Vinyals et al., 2019), and board games like Chess, Go, or Shogi (Silver et al., 2017). However, these advances can be attributed to fine-tuned agents each trained to solve the specific task. For example, a robot trained to hit a baseball would not be able to play table tennis, even though both tasks involve swinging at a ball. If we were to train the agent with current learning-based methods on a new task, then it would tend to forget previous tasks and skills.

By contrast, humans continuously learn, remembering many tasks and using past experiences to help learn new tasks in different environments over extended periods of time. Developing agents capable of continuously building on what was learned previously, without forgetting knowledge obtained from the past, is crucial in order to deploy robotic agents into everyday scenarios. This capability is referred to as *continual learning* (Ring, 1994; 1998), also known as *never-ending learning* (Mitchell et al., 2015; Chen et al., 2013), *incremental learning*, and *lifelong learning* (Thrun & Mitchell, 1995).

In recent years, there has been a growing interest in building agents that continuously learn skills or tasks without forgetting previously learned behavior (Khetarpal et al., 2020). However, unlike other areas leveraging machine learning such as computer vision and natural language processing, growth in the field of continual RL is still quite limited. Why is that? We argue there are three primary reasons, (a) missing code: research code in continual RL is not publicly available (due to the use of proprietary code for training agents at scale) and reproducibility remains difficult. This creates a high barrier to entry as any new entrant must re-implement and tune baselines, in addition to designing their own algorithm; (b) high compute barrier: the lack of publicly released baselines is compounded by the fact that extensive compute resources are required to run experiments used in prior work, which is not easily available in

---

\* Equal contribution

academic settings; (c) benchmarking gaps: there are few benchmarks and metrics used to evaluate continual RL, with no set standards.

With this work, we aim to democratize the field of continual RL by reducing the barriers to entry and enable more research groups to develop algorithms for continual RL. To this end, we introduce CORA, a platform that includes benchmarks, baselines, and metrics for **Continual Reinforcement Learning Agents**. With this platform, we present three contributions to the field which we believe will support progress towards this goal. First, we present a set of benchmarks, each tailored to measure progress toward a different goal of continual learning. Our benchmarks include task sequences designed to: test generalization to unseen environment contexts (Procgen), evaluate scalability to the number of tasks being learned (MiniHack), and exercise scalability in realistic settings (CHORES), in addition to a standard, proven benchmark (Atari). Second, we provide three metrics to compare key attributes of continual RL methods on these benchmarks: Continual Evaluation, Isolated Forgetting, and Zero-Shot Forward Transfer. Finally, we release open-source implementations of previously proposed continual RL algorithms in a shared codebase, including CLEAR (Rolnick et al., 2019), a state-of-the-art method. We demonstrate that while CLEAR outperforms other baselines in Atari and Procgen, there is still significant room for methods to improve on our benchmarks.

In all, we present the CORA platform<sup>1</sup>, designed to be a modular and extensible code package which brings all of this together. We hope our platform will be a one-stop shop for developing new algorithms, comparing to existing baselines using provided metrics, and running methods on evaluations suitable for testing different aspects of continual RL agents. We believe our contributions will help researchers conveniently develop new continual RL methods and facilitate communicating research results to the community in a standard fashion. If continual robot learning can successfully utilize household benchmarks like CHORES, then household robots or robots in the workforce may not be far distant.

## 2 RELATED WORK

We discuss various environments and tasks used to benchmark reinforcement learning in Appendix A.

**Evaluating continual reinforcement learning** While continual learning is most commonly addressed in the context of supervised learning for image classification such as in (Ruvolo & Eaton, 2013; Lopez-Paz & Ranzato, 2017; Hsu et al., 2018; Javed & White, 2019; Hsu et al., 2018; Mai et al., 2021), here we focus our discussion on continual reinforcement learning, and as such, simulation environments and tasks to benchmark RL agents. For an overview on continual learning applied to neural networks in general, we refer the reader to Parisi et al. (2019) and Mundt et al. (2020).

With CORA, we introduce benchmarks designed to evaluate continual RL algorithms that can be used in more challenging, realistic scenarios. Continual RL for policies or robotic agents (Lesort et al., 2020) is more nascent, although several benchmarks have been proposed. As mentioned above, continual RL has typically been evaluated on a sequence of Atari games (Kirkpatrick et al., 2017; Schwarz et al., 2018b; Rolnick et al., 2019) and we leverage these prior results to validate our baselines. Other video game-like environments proposed to evaluate continual learning include StarCraft (Schwarz et al., 2018a) and VizDoom (Lomonaco et al., 2020).

Procgen (Cobbe et al., 2020) and MiniHack (Küttler et al., 2020; Samvelyan et al., 2021), two of our other benchmarks, are procedurally-generated, like Jelly Bean World (Platanios et al., 2020) which is a procedurally-generated 2D gridworld proposed as a testbed for continual learning agents. Nekoei et al. (2021) present *Lifelong Hanabi* for continual learning in a multi-agent RL environment. Beyond game-like environments, Wołczyk et al. (2021) evaluate continual RL using task boundaries in a multi-task robot manipulation environment, while Khetarpal et al. (2018) discuss home simulations as a potentially suitable environment to benchmark continual RL. In this work we present CHORES in AI2-THOR (Kolve et al., 2017): task sequences for an agent in a home simulation to evaluate continual RL methods in the visually realistic scenes offered.

Our work is conceptually similar to bsuite (Osband et al., 2019), which curates a collection of toy, diagnostic experiments to evaluate different capabilities of a standard, non-continual RL agent. Concurrent with our work, Sequoia (Normandin et al., 2021) introduces a software framework with baselines, metrics, and evaluations aimed at unifying research in continual supervised learning and continual reinforcement learning. While both are valuable benchmarking tools, they focus predominantly on simpler tasks like MNIST and CartPole. The most complex environments that Sequoia uses are Meta-World, with simple state-based manipulation tasks, and MonsterKong, composed of 8 hand-designed platformer levels. In contrast to both, CORA presents challenging task sequences for vision-based, procedurally-generated environments that evaluate generalization and scalability for continual RL. Also concurrent with our work, Avalanche RL (Lucchesi et al., 2022) introduces a library for continual RL, whose functionality will be merged into Avalanche (Lomonaco et al., 2021), is a popular library for continual learning.

---

<sup>1</sup>[https://github.com/AGI-Labs/continual\\_rl](https://github.com/AGI-Labs/continual_rl)

However, Avalanche RL does not present any experimental results on baseline methods. In this paper, we evaluate several continual RL methods across four different environments.

### 3 TASK SEQUENCES FOR BENCHMARKING CONTINUAL RL

The goal of continual reinforcement learning is to develop an agent that can learn a variety of different tasks in non-stationary settings. To this end, prior work has primarily focused on preventing catastrophic forgetting (Kirkpatrick et al., 2017; Rolnick et al., 2019; Schwarz et al., 2018b) and maintaining plasticity (Mermilliod et al., 2013) so that the agent can learn new tasks. While simple tasks are useful for debugging, skill on them does not necessarily translate to more complex tasks. We believe that the field has matured enough and is ready for more ambitious goals. In particular, we believe continual RL methods should address the following problems: (a) showing positive forward transfer by leveraging past experience; (b) generalizing to unseen environment contexts; (c) learning similar tasks through provided goal specifications; (d) improving sample efficiency; in addition to (e) mitigating catastrophic forgetting; and (f) maintaining plasticity.

While a single benchmarking environment that suitably deals with each of these features may be ideal, over the course of development we have found this to be impractical with the tools currently available. For example, visually-realistic, physics-based environments are generally not fast enough for the longer sequences of tasks that we use to test resilience to forgetting. Furthermore, it may be overbearing for new algorithms to sufficiently address every continual RL goal, whereas a modular set of evaluations allows for researchers to focus on areas to best highlight particular contributions of their new methods. Instead, we present four benchmarks which continual RL researchers may utilize:

- Atari (Bellemare et al., 2013), 6 task sequence: A standard, proven benchmark used by Schwarz et al. (2018b) and Rolnick et al. (2019), particularly to demonstrate resilience to catastrophic forgetting.
- Procgen (Cobbe et al., 2020), 6 task sequence: Designed to test resilience to forgetting and in-distribution generalization to unseen contexts in procedurally-generated, visually-distinct environments.
- MiniHack (Küttler et al., 2020), 15 task sequence, based on NetHack (Samvelyan et al., 2021): Designed to train agents on a long sequence of tasks in environments that are stochastic, procedurally-generated, and visually-similar, in order to demonstrate resilience to forgetting, maintenance of plasticity, forward transfer, and out-of-distribution generalization (extrapolation along different environment factors).
- 4 different CHORES, utilizing ALFRED (Shridhar et al., 2020) and AI2-THOR (Kolve et al., 2017): Designed to test agents in a visually realistic domain where sample efficiency is key. Unlike other environments where different tasks may be easily identified visually, CHORES tasks explicitly provide a goal image. CHORES also present an opportunity to test forward transfer due to task similarity. For example, the ability to pick up a hand towel ideally should transfer from one bathroom to another.

We direct the reader to Appendix B for formalism and background on the continual RL setting, including more precise definitions of generalization for these benchmarks and how continual RL applies to these task sequences.

Task selection and ordering is still an open area of research (Jiang et al., 2021), and we did not tune task ordering. We use the Atari task sequence as demonstrated in prior work, as well as use the implicit ordering in which Procgen and MiniHack presented their tasks. Selection of tasks for CHORES was more involved, and is described in Appendix C.1.

Our goals include reducing the compute costs of continual RL experiments for the new benchmarks, as compared to the Atari experiments. Indeed, we observed a speedup of 7x for MiniHack, 6x for Procgen, and 2x for CHORES; details are available in Appendix C.7.

#### 3.1 ATARI TASKS

Building off the work of Kirkpatrick et al. (2017) that evaluated a random set of ten Atari (Bellemare et al., 2013) games, recent work in continual reinforcement learning (Schwarz et al., 2018b; Rolnick et al., 2019) evaluates continual learning on six Atari games: [0-SpaceInvaders, 1-Krull, 2-BeamRider, 3-Hero, 4-StarGunner, 5-MsPacman]. They train agents on each of the six tasks for 50M frames, cycling through the sequence 5 times, for a total of 1500M frames seen. This results in 250M frames per task, which is five times as many frames as is standard in the single task setting. The primary focus of algorithms that were developed and evaluated on this Atari task sequence was to reduce catastrophic forgetting. This setting is particularly suitable for catastrophic forgetting due to the lack of overlap between tasks, in regards to both observations and skills required. More details are given in Appendix C.8.

Following (Kirkpatrick et al., 2017; Schwarz et al., 2018b; Rolnick et al., 2019), we use the original Atari settings, meaning that the games are deterministic. Modifications such as “sticky actions” (Machado et al., 2018) have become

more standard to overcome simulator determinism, and are an option for increasing task difficulty in future work. In this work, we use Atari to validate our baseline implementations, preferring procedurally-generated environments (Procgen, MiniHack) to test generalization. We note that different Atari game modes may also be used to produce variation and assess generalization capability, as proposed by [Farebrother et al. \(2018\)](#).

### 3.2 PROCGEN TASKS

We use Procgen ([Cobbe et al., 2020](#)) to define a new sequence of video game tasks, with the intention of replacing the Atari tasks used previously to evaluate continual learning methods. We choose Procgen because its procedural generation allows for evaluating generalization on unseen levels, unlike Atari. Like Atari however, the tasks are all visually distinct and the task sequence is well-suited to evaluating catastrophic forgetting. As with Atari, this is due to a general lack of overlap between tasks. From the full set of available Procgen environments, the specific set of tasks was chosen by [Igl et al. \(2021\)](#) to ensure the existence of a nontrivial generalization gap and to ensure generalization actually improves during training.

Procgen is also significantly faster to run (our experiments take several days for Procgen vs. weeks on Atari). To improve sample efficiency and reduce compute costs, we use the easy distribution mode for these Procgen games. We use a sequence of six tasks [0-Climber, 1-Dodgeball, 2-Ninja, 3-Starpilot, 4-Bigfish, 5-Fruitbot], training for 5M frames on each task with 5 learning cycles. This results in 25M frames per task and 125M frames total. Note that we are not increasing the number of training frames per task compared to the original paper, unlike the Atari task sequence.

The observation space is (64, 64) RGB images and is not framestacked. The 15-dim action space is the same across Procgen tasks. As recommended in the original paper, we train the agent on 200 levels, while evaluation uses the full distribution of levels that Procgen can procedurally generate. What is randomized varies depending on the game environment, but covers textures, enemies, objects, and room layouts. See Appendix C.4, Figure 7 for a visualization of the observations an agent may receive for each task in Procgen.

### 3.3 MINIHACK’S NETHACK TASKS

Most prior continual RL work evaluates on a relatively small number of tasks, but the recently introduced MiniHack ([Samvelyan et al., 2021](#)) environment is fast enough to enable scaling up. MiniHack is based on the NetHack Learning Environment ([Küttler et al., 2020](#)), a setting that is procedurally generated like Procgen and has stochastic dynamics (such as when attacking monsters). As with Procgen, the variation over which levels are randomized differs by environment, but includes objects, enemies, start & goal locations, and room layouts. The larger number of tasks enables MiniHack to more extensively test an agent’s ability to prevent forgetting and to maintain plasticity. Additionally, while the Procgen tasks are easy to tell apart visually, the MiniHack tasks use the same texture assets and are more challenging to distinguish. This makes task identification and boundary detection more difficult.

To create the MiniHack task sequence, we define 15 (train, test) task pairs with a total of 27 different navigation-type tasks. The training environments are the easier versions, and we evaluate the agent on the harder environment variant. We select from the navigation-type tasks introduced by MiniHack, ordering by how MiniHack presents their tasks, and only omit the tasks that require episodic memory and deep exploration. We provide the full MiniHack task sequence we use in Appendix C.3. Three evaluation environments are each used twice, because each has two related training tasks, the impact of which we discuss further in Section 6.2. MiniHack also provides skill acquisition tasks, which could be used in future work for an even more challenging task sequence.

When reporting results on this task sequence in Figure 3, we use the training environment name to refer to each task. We use only the pixel-based input for the agent. MiniHack renders an (80, 80) RGB image which we zero-pad to (84, 84) for convenience. See Appendix C.4, Figure 7 for a visualization of observations an agent may receive for each task in MiniHack. All tasks share an 8-dim action space.

### 3.4 CHORES BENCHMARK SUITE USING ALFRED AND AI2-THOR

AI2-THOR ([Kolve et al., 2017](#)) is a visually realistic simulation environment that provides a variety of rooms for an agent to act in, with 30 layouts each of bedrooms, living rooms, kitchens, and bathrooms. ALFRED ([Shridhar et al., 2020](#)) is a benchmark for embodied vision-and-language agents which provides demonstrations for extended sequences of complex, tool-based tasks defined using AI2-THOR.

Using the demonstration trajectories and task definitions from ALFRED, we define a set of environments and task sequences for continual RL, which we refer to as **Continual Household Robot Environment Sequences (CHORES)**. We do not provide ALFRED demonstration trajectories to the agent, as learning from demonstrations is beyond the



Figure 1: Examples for CHORES that show the variation within each task sequence.

scope of this paper. Instead, we leverage the demonstration data to initialize an AI2-THOR environment and generate subgoal images for the agent, which communicate the intended task for the agent to perform. The initial state of the environment is set to the initial state of the demonstration trajectory. The usage of these demonstrations enables us to have a variety of initializations for robot location, object locations, and room instance without explicitly setting the simulation parameters or hand-defining distributions over these parameters. ALFRED also defines reward functions for its tasks based on achieving its subgoals, which we use. Figure 5 in Appendix C.1 visualizes an example of a full set of CHORES subgoals.

Our CHORES benchmark extends continual RL into a visually realistic domain, where sample efficiency is key and where tasks bear similarities that make forward transfer particularly useful. Sample efficiency is critical because we designed CHORES to use a tight frame budget, as an initial attempt to mirror what would be feasible in the real world.

We first define three CHORES that shift the environment context in well-defined ways: **Mem-VaryRoom** changes the room scene, **Mem-VaryTask** changes the task type, and **Mem-VaryObject** changes the object with which the agent interacts. The fourth CHORES, **Gen-VaryTraj**, is considerably harder than the first three: it varies both the object and the scene, in addition to testing generalization on unseen contexts from heldout demo trajectories. Figure 1 visualizes CHORES and shows examples of variation within each task sequence. In Appendix C.1, we discuss design objectives and compute constraints used while creating the set of CHORES we introduce in this work. Appendix C.2 describes the four CHORES proposed in more detail. We note that the CHORES protocol is not exclusive to AI2-THOR and can also be applied using any home simulation with a diverse dataset of demonstrations.

We use an action space of 12 discrete actions (e.g. LookDown, MoveAhead, SliceObject, PutObject, etc.). For an action that interacts with an object, we take the action with the correct task-relevant object. Note that this differs from agents evaluated in ALFRED originally, which generate interaction masks to select one object from those in view to interact with. We use an observation size of (64, 64, 6), with 3 channels for the current RGB image observation and 3 channels for an RGB goal image.

## 4 METRICS

We refer the reader to Appendix B for background and full review of the continual RL setting. We assume  $N$  tasks are presented as a sequence  $\mathcal{S}_N := (\mathcal{T}_0 \dots \mathcal{T}_{N-1})$ . The agent trains on task  $\mathcal{T}_i$  at timesteps in the interval  $[A_i, B_i]$ , where  $A_i$  and  $B_i$  are the task boundaries denoting the start and end, respectively, of task  $\mathcal{T}_i$ . We cycle through the tasks  $M$  times, so the full task sequence  $\mathcal{S}_{NM}$  has length  $N \cdot M$ .

At each timestep  $t$ , the policy  $\pi$  receives an observation, reward, and indicator of whether the episode is done, and takes an action based on what it has observed. In this section, we define episode return as the undiscounted sum of rewards received over an episode. We train the agent  $s$  different times on the task sequence, with each run using a different initial random seed. We consider several expected episode returns to be used when defining our metrics:

$$r_{i,\cdot,t=K} := \mathbf{R}_{t=K}(\pi, \mathcal{T}_i) \quad \text{expected return achieved on task } \mathcal{T}_i \text{ at timestep } K \quad (1)$$

$$r_{i,j,end} := \mathbf{R}_{t=B_j}(\pi, \mathcal{T}_i) \quad \text{expected return achieved on task } \mathcal{T}_i \text{ after training on task } \mathcal{T}_j \quad (2)$$

$$r_{i,all,max} := \max_{K \in [A_0, B_{N-1}]} \mathbf{R}_{t=K}(\pi, \mathcal{T}_i) \quad \begin{matrix} \text{maximum (over all timesteps) expected return achieved on task } \mathcal{T}_i \\ \text{after training on all tasks} \end{matrix} \quad (3)$$

Using these definitions, we discuss metrics for measuring different attributes of continual RL agents. Before proceeding, we describe how we estimate expected returns. A run involves training one instance of an agent on a task sequence. We pause each run every  $n$  timesteps and evaluate  $E = 10$  episodes worth of data for every task in the sequence, and record the means. We further smooth the returns by using a moving average with a rolling window of size  $w$  to get an estimate of  $r$ .

#### 4.1 CONTINUAL EVALUATION, ISOLATED FORGETTING, AND ZERO-SHOT FORWARD TRANSFER

We use three metrics to evaluate the performance of an algorithm on our benchmarks. The first is the standard Continual Evaluation metric as used by prior work (Rolnick et al., 2019; Schwarz et al., 2018b). The second metric, Isolated Forgetting, measures how much an agent may forget an old task while learning a new task. The third metric, Zero-Shot Forward Transfer, measures how much an old task may contribute to the learning of a new task.

Continual Evaluation presents the opportunity to compute our definitions of Forgetting and Transfer, which isolate the effects that training on task  $\mathcal{T}_j$  has on the performance of task  $\mathcal{T}_i$ , without requiring single-task models to be trained and evaluated separately, as in (Chaudhry et al., 2018). We present our Forgetting and Transfer metrics in two forms. The first form is the summary statistic, which provides a high-level overview of performance and is shown for our benchmarks in Table 1. The second form is as a diagnostic table that describes how training on task  $\mathcal{T}_j$  (column) impacts each task  $\mathcal{T}_i$  (row). Appendix E contains the full diagnostic tables for all benchmarks.

**Continual Evaluation ( $\mathcal{C}$ ):** The Continual Evaluation metric, presented as a set of graphs, evaluates performance on all tasks periodically during training. This provides an understanding at any point of how the agent performs on every task  $\mathcal{T}_i$  at timestep  $K$ . This is essentially a Monte Carlo estimate,  $\mathcal{C}_i(t = K) \approx r_{i, \cdot, t=K}$ .

While the agent is asked to pause for evaluation every  $n$  timesteps, in practice there is some variation due to the asynchronous implementation of the agents. To align the evaluation data across runs, we linearly interpolate to a common interval, then compute the mean and standard error over  $s$  seeds. We graph  $\mathcal{C}_i$  for each task; an example can be seen in Figure 2. We also provide the final performance mean and standard error for each method in a table format, for convenient reference.

**Isolated Forgetting ( $\mathcal{F}$ ):** Isolated Forgetting, originally inspired by (Lopez-Paz & Ranzato, 2017; Wang et al., 2021), represents how much is forgotten from a learned task during later tasks. It compares the expected return achieved for earlier task  $\mathcal{T}_i$  before and after training on later task  $\mathcal{T}_j$ , where  $i < j$ :

$$\mathcal{F}_{i,j} = \frac{r_{i,j-1,end} - r_{i,j,end}}{|r_{i,all,max}|} \quad (4)$$

When  $\mathcal{F}_{i,j} > 0$ , the agent has become worse at past task  $\mathcal{T}_i$  while training on new task  $\mathcal{T}_j$ , indicating forgetting has occurred. Conversely, when  $\mathcal{F}_{i,j} < 0$ , the agent has become better at task  $\mathcal{T}_i$ , indicating *backward transfer* (Lopez-Paz & Ranzato, 2017) has been observed. We normalize by the absolute value of the maximum expected return observed for task  $\mathcal{T}_i$  within the run. Tasks can have varying reward scales, and normalization helps for comparing between tasks.

Unlike Chaudhry et al. (2018), we do not use the max value observed, but rather the value right before training on a task, which corresponds with Lopez-Paz & Ranzato (2017). This allows us to isolate the impact of that particular task, rather than looking at the cumulative effect to that point. We believe this makes the results easier to understand overall, and the metric more useful.

**Zero-Shot Forward Transfer ( $\mathcal{Z}$ ):** Forward transfer considers how much prior tasks aid in the learning of new tasks. The Intransigence metric as defined by Chaudhry et al. (2018) measures forward transfer by comparing the maximum expected return for a task trained independently to the expected return achieved while it was trained sequentially. While this might be the most accurate way to evaluate forward transfer, computing independent performance effectively doubles the amount of compute required.

As one of our goals is to minimize the compute requirements of this benchmark, we instead propose what we refer to as the Zero-Shot Forward Transfer metric. It compares the expected return achieved for later task  $\mathcal{T}_i$  before and after training on earlier task  $\mathcal{T}_j$ , where  $i > j$ :

$$\mathcal{Z}_{i,j} = \frac{r_{i,j,end} - r_{i,j-1,end}}{|r_{i,all,max}|} \quad (5)$$

When  $\mathcal{Z}_{i,j} > 0$ , the agent has become better at later task  $\mathcal{T}_i$  having trained on earlier task  $\mathcal{T}_j$ , indicating forward transfer has occurred by *zero-shot* learning (Lopez-Paz & Ranzato, 2017; Díaz-Rodríguez et al., 2018). When  $\mathcal{Z}_{i,j} < 0$ , the agent has become worse at task  $i$ , indicating negative transfer has occurred. We normalize by the absolute value of the maximum expected return observed for task  $\mathcal{T}_i$  within the run, as was done for Forgetting.

**Additional details** (a) We only consider one cycle of the task sequence to compute  $\mathcal{F}$  and  $\mathcal{Z}$ . We do this for interpretability and ease of understanding, though these metrics could be averaged across cycles with additional assumptions. (b) The diagnostic tables and summary statistics are given using the unseen testing environments when available. We use the same task indices when referring to the training and testing environments. (c) We scale

by 10 for readability<sup>2</sup>, and average across seeds. (d) Summary statistics are computed as an average across tasks:  $\bar{\mathcal{F}} = \sum_{i < j} \frac{1}{s} \sum_s \mathcal{F}_{i,j}$  and  $\bar{\mathcal{Z}} = \sum_{i > j} \frac{1}{s} \sum_s \mathcal{Z}_{i,j}$ . Additionally, we compute the standard error of the mean; details are given in Appendix E.1.

**Why use offline evaluation?** While a continual agent operating in the real world would not pause for evaluation, offline evaluation is a useful tool enabled by simulation to understand agent performance. Offline evaluation helps answer questions such as: “How does the agent currently perform on tasks learned in the past?”, “How does experience the agent has acquired help it learn new tasks?”, or “How well would the agent generalize if it were asked to perform the task in a unseen environment?”.

## 5 CORA: A PLATFORM FOR CONTINUAL REINFORCEMENT LEARNING AGENTS

### 5.1 BASELINES

We re-implemented four continual RL methods with baseline results on the Atari sequence from Section 3.1, which are not publicly available to the best of our knowledge. We prioritized methods which had been demonstrated on Atari before, in order to reference such results and appropriately validate our implementations. The continual RL methods were selected to cover the categorizations described by Parisi et al. (2019); Lesort et al. (2020); Mundt et al. (2020): elastic weight consolidation (EWC) (Kirkpatrick et al., 2017), online EWC (Schwarz et al., 2018b), Progress and Compress (P&C) (Schwarz et al., 2018b), Continual Learning with Experience and Replay (CLEAR) (Rolinck et al., 2019). EWC is a Regularization approach, P&C is an Architectural approach, and CLEAR is a Rehearsal approach. Note that CLEAR is task-agnostic, while EWC and P&C require explicit task boundaries.

Our implementations of these baselines all build off the IMPALA (Espeholt et al., 2018) architecture and use the open-source TorchBeast code (Küttler et al., 2019). We discuss implementation details in Appendix C.5. In Appendix C.8, we validate the performance of our baseline implementations compared to that of the original implementations on Atari.

### 5.2 CODE PACKAGE

We release our `continual_rl` codebase<sup>3</sup> as a convenient way to run continual RL baselines on the benchmarks we outlined in Section 3 and to use the continual RL evaluation metrics we defined in Section 4. The package is designed modularly, so any component may be used separately elsewhere, and new benchmarks or algorithms may be integrated in. We provide more details on design and usage of the `continual_rl` package in Appendix D. Hyperparameters for all experiments are made available as configuration files in the codebase, and also detailed in Appendix C.6.

	IMPALA	EWC	Online EWC	P&C	CLEAR		IMPALA	EWC	Online EWC	P&C	CLEAR
Atari	2.3 ± 0.1	0.3 ± 0.3	1.6 ± 0.1	1.8 ± 0.1	0.7 ± 0.1		0.1 ± 0.0	0.1 ± 0.2	-0.0 ± 0.0	-0.0 ± 0.1	0.0 ± 0.0
Procgen	1.2 ± 0.0	0.7 ± 0.0	1.1 ± 0.0	0.5 ± 0.0	-0.0 ± 0.0		-0.1 ± 0.0	-0.2 ± 0.1	-0.1 ± 0.1	0.1 ± 0.1	-0.1 ± 0.0
MiniHack	0.3 ± 0.0	-	-	-	0.1 ± 0.0		0.6 ± 0.0	-	-	-	0.5 ± 0.1
C-VaryRoom	-	-0.6 ± 0.6	-	0.0 ± 0.0	-1.7 ± 1.7		-	-0.0 ± 0.0	-	3.2 ± 1.9	-1.1 ± 1.1
C-VaryTask	-	2.1 ± 1.4	-	-2.2 ± 2.2	1.5 ± 0.2		-	4.0 ± 2.6	-	0.2 ± 0.1	-3.2 ± 0.0
C-VaryObj	-	-1.0 ± 1.1	-	2.0 ± 2.0	3.4 ± 0.2		-	2.6 ± 2.9	-	5.4 ± 1.3	-4.6 ± 0.8
C-MultiTraj	-	0.7 ± 1.2	-	-0.1 ± 0.1	-0.3 ± 2.1		-	4.0 ± 0.5	-	0.4 ± 0.1	-4.7 ± 0.7

(a) Forgetting ( $\mathcal{F}$ ) summary statistics for all experiments.

(b) Transfer ( $\mathcal{Z}$ ) summary statistics for all experiments.

Table 1: Summary statistics for all benchmarks and for all methods evaluated on them.

## 6 EXPERIMENTAL RESULTS

In this section, we present results on Procgen (Section 6.1), MiniHack (Section 6.2), and CHORES (Section 6.3). **For Atari results, see Appendix C.8.** Metric summary statistics for all methods can be seen in Table 1, and metric diagnostic tables are available in Section E. Final performance tables are also available in Appendix C.10, Tables 5 through 10. To estimate expected return and compute metrics, we use the following values for parameters described in

<sup>2</sup>For instance, “4.2” uses 1 less character than “0.42”, so this scaling helps fit the metric tables horizontally in the paper.

<sup>3</sup>Our code: [https://github.com/AGI-Labs/continual\\_rl](https://github.com/AGI-Labs/continual_rl)

Section 4: Procgen:  $n = 0.25e6$ ,  $w = 20$ ,  $s = 20$ ; MiniHack:  $n = 1e6$ ,  $w = 20$ ,  $s = 10$ , CHORES:  $n = 5e4$ ,  $w = 5$ ,  $s = 3$ , Atari:  $n = 0.25e6$ ,  $w = 20$ ,  $s = 5$ .

On the Continual Evaluation plots, solid lines represent evaluation on unseen testing environments, while dashed lines show evaluation on the training environments. Shaded grey rectangles are used to indicate which task is being trained during the indicated interval. We plot the mean as each line and the standard error as the surrounding, shaded region.

In each Forgetting table, we show negative values (representing backwards transfer) in green and positive in red, darker in proportion to the magnitude of  $\mathcal{F}$ . Values close to zero are unshaded. In contrast, in each Transfer table, we show positive values (indicating forward transfer) in green shades and negative values in red.

We proceed to discuss experimental results with CORA using two perspectives. First, from the viewpoint of benchmark analysis, we empirically discuss what each benchmark is evaluating and give examples of how the metric tables may be used. Second, from the view of algorithm design, we examine the performance of the baselines to identify axes which can be improved on by future algorithms. We frame this section through these two lenses in order to show how CORA may be used by end-users.

## 6.1 PROCGEN RESULTS

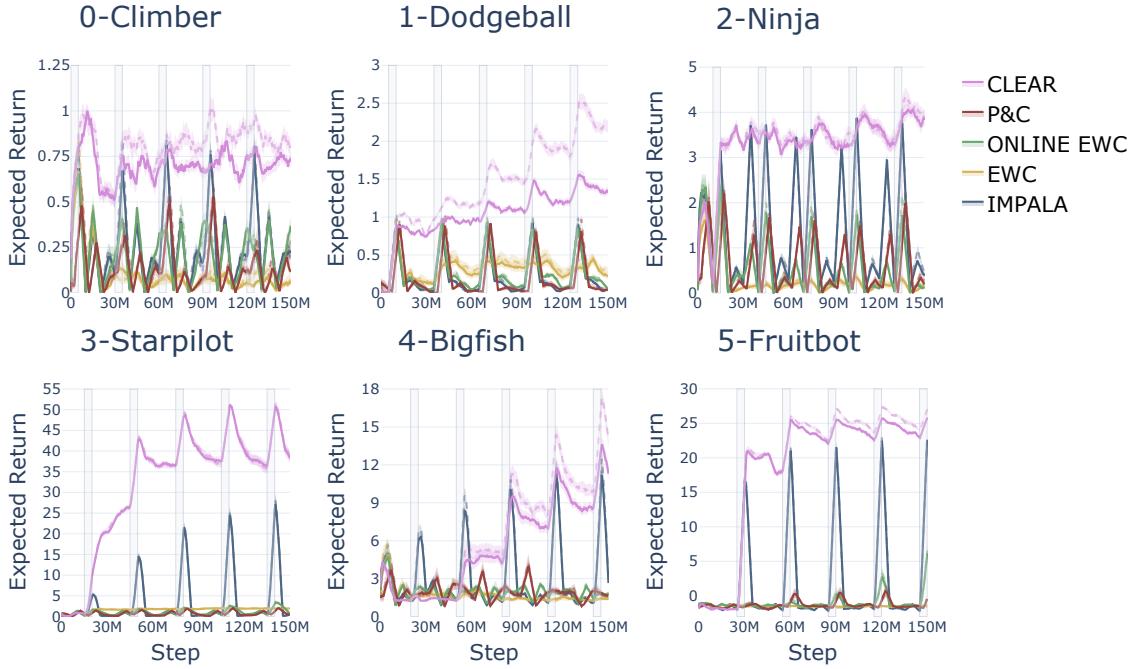


Figure 2: Results for Continual Evaluation ( $\mathcal{C}$ ) on the 6 Procgen tasks, based on recommendations by (Igl et al., 2021). The solid line shows evaluation on unseen testing environments; the dashed line shows evaluation on training environments. Gray shaded rectangles show when the agent trains on each task. For readability, Figure 10 in Appendix C.9 presents an alternative version of this figure without CLEAR results.

**Benchmark analysis:** From the summary statistics in Table 1, we can see that Procgen tests for catastrophic forgetting, but shows little forward transfer overall, which aligns with our expectations for this benchmark. Using the Transfer metric diagnostic tables in Appendix E.5, we see that that forward transfer is not uniform across tasks. For example, we observe that 0-Climber transfers reasonably well to 2-Ninja and 4-Bigfish. Intuitively, as 0-Climber and 2-Ninja are both platformer games, transfer is expected. Transfer to 4-Bigfish is less obvious but may be explained by both games using side-view perspectives or by sharing useful skills like object gathering. In particular, 0-Climber involves collecting stars, while 4-Bigfish tasks the agent with eating other fish.

**Algorithm design:** From the Continual Evaluation results in Figure 2, we observe that CLEAR is a strong baseline for avoiding catastrophic forgetting on all tasks, reliably outperforming every other method. However, there is still room for improvement: maximum scores obtained by CLEAR fall significantly short of the maximum achievable scores reported in Appendix C of the Procgen paper (Cobbe et al., 2020), particularly on 0-Climber (1 vs 12.6), 1-Dodgeball (2.5 vs 19),

2-Ninja (4 vs 10), and 4-Bigfish (18 vs 40). Additionally, by comparing the training (dashed) and testing (solid) lines, we observe that CLEAR generalizes well to unseen contexts on all tasks, except 1-Dodgeball. The summary statistics in Table 1 show that transfer is overall low for Procgen. Using the more detailed diagnostic tables in Appendix E.5, we can see that this varies by task. For instance with EWC, training on 1-Dodgeball improves performance on 3-Starpilot but reduces performance on all other tasks. CLEAR shows some transfer from 0-Climber to 2-Ninja, but essentially none anywhere else, even showing negative transfer from 1-Dodgeball to 2-Ninja. These failures represent opportunities for investigation and for new algorithms to improve on.

## 6.2 MINI HACK RESULTS

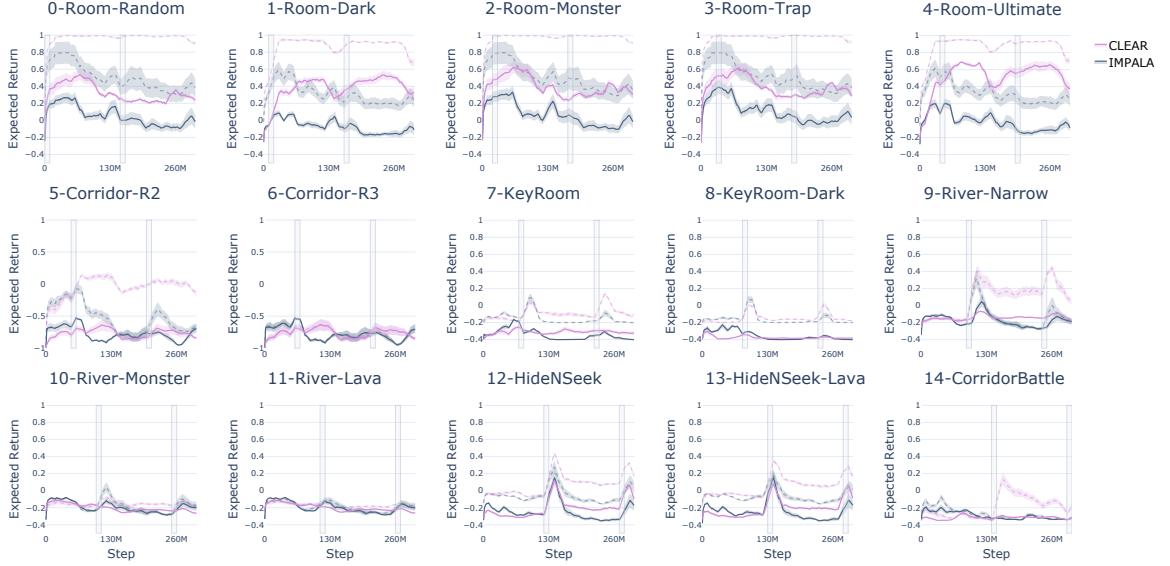


Figure 3: Results for Continual Evaluation ( $\mathcal{C}$ ) on the 15 MiniHack task pairs sequence. The solid line shows evaluation on unseen testing environments; the dashed line shows evaluation on training environments. Gray shaded rectangles show when the agent trains on each task.

**Benchmark analysis:** From the summary statistics in Table 1, we observe that IMPALA and CLEAR show minor transfer on MiniHack tasks compared to Atari and Procgen. Looking at the Transfer metric diagnostic table in Appendix E.7, we can see that the first task 0-Room-Random transfers significantly to all other tasks, which can be interpreted as the agent learning the basics of moving around a MiniHack environment. Furthermore, we observe transfer from environments to others of the same type. For example, the Room environments generally positively transfer to each other, while mostly negatively transferring to the later River and HideNSeek environments. When training tasks share the same testing task, such as 12-HideNSeek and 13-HideNSeek-Lava, the transfer metric is noticeably high, as expected. Finally, from the Continual Evaluation results in Figure 3, we observe that MiniHack effectively tests for plasticity as well, as later experiments fail to learn effectively.

**Algorithm design:** From the Continual Evaluation results in Figure 3, we can see CLEAR generally performs well at learning tasks and mitigating catastrophic forgetting for the first five tasks. However, we observe that the agent struggles to learn later tasks (fails to maintain plasticity), and that there is a significant out-of-distribution generalization gap, in performance on test (solid) compared to train (dashed) environments for all tasks. Additionally, inspecting the forgetting metric diagnostic tables in Appendix E.6, we see that the HideNSeek tasks exhibit particularly high forgetting. These results and shortcomings present important areas for new algorithms to pursue.

## 6.3 CHORES RESULTS

We show Continual Evaluation results for the four CHORES in Figure 4. Forgetting and Transfer metrics diagnostic tables are available in Appendix E.8 and Appendix E.9. On the memorization sequences, the testing environment is the same as the training environment, and evaluation is represented with a solid line. In our generalization experiment, the solid line represents evaluation on held-out testing environments, and the dotted line represents performance on the

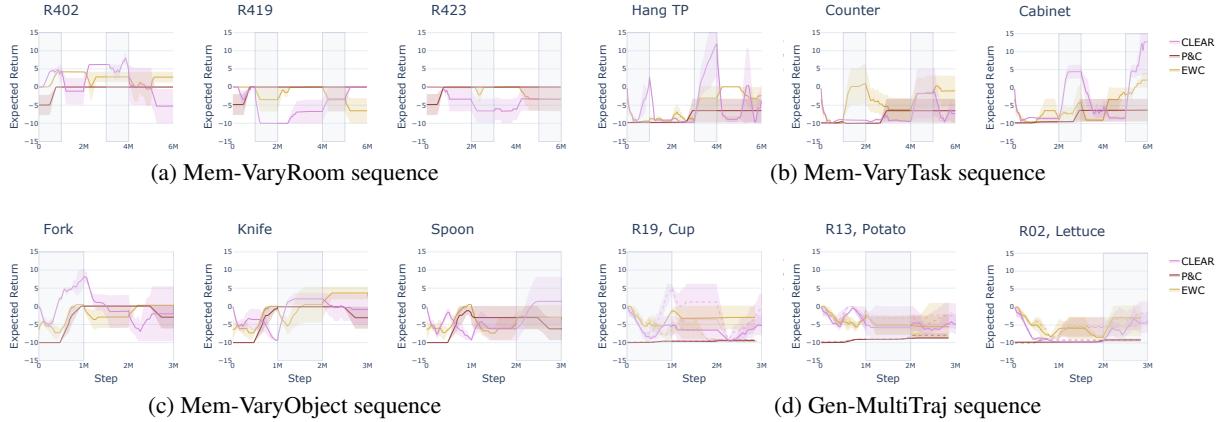


Figure 4: Results for Continual Evaluation ( $\mathcal{C}$ ) on the CHORES suite of benchmarks. For (d) Gen-MultiTraj, the solid line shows evaluation on unseen testing environments; the dashed line shows evaluation on training environments. Gray shaded rectangles show when the agent trains on each task.

training environments. We report 2 cycles for each of (a) Mem-VaryRoom and (b) Mem-VaryTask, and 1 for each of (c) Mem-VaryObject and (d) Gen-MultiTraj, due to time constraints.

**Benchmark analysis:** From the Continual Evaluation results, we find that CHORES are challenging, and current agents achieve low returns overall. Some learning occurs for the first task of every sequence, but nearly none in later tasks, with the exception of (a) Mem-VaryTask. We also observe no generalization to unseen contexts from held-out demo trajectories in (d) Gen-MultiTraj. The Transfer and Forgetting metrics are less meaningful with low returns, but there is some indication of forward transfer, particularly on (c) Mem-VaryObj. Taken together, we can see CHORES as the reach-goal; a set of tasks current methods cannot solve, and that will truly test the sample efficiency of future methods.

**Algorithm design:** CLEAR achieves the highest returns overall, but there is significant room to improve learning these tasks. We observe significant Forgetting, particularly on (c) Mem-VaryObj, illustrating one such area for improvement. Advances in sample efficiency and exploration are likely required for agents to make progress on this challenge.

## 7 CONCLUSION

In this paper, we present CORA, a platform designed to reduce the barriers to entry for continual reinforcement learning. CORA provides a set of benchmarks, open-sourced implementations of several baselines, evaluation metrics, and the modular `continual_rl` package to contain it all. Each benchmark is designed to exercise different aspects of continual RL agents: a standard, proven Atari benchmark for catastrophic forgetting and sample efficiency; a Procgen benchmark to test forgetting and generalization to unseen environment contexts; a MiniHack benchmark to test generalization, plasticity, and transfer; and the new, challenging, CHORES benchmark to test capability in a visually-realistic environment where sample-efficiency is key. With these benchmarks, we demonstrate the strengths and weaknesses of the current state-of-the-art continual RL method, CLEAR. While CLEAR generally outperforms the other baselines at learning tasks and mitigating catastrophic forgetting, significant improvements are needed for generalization, forward transfer, and maintaining plasticity over a long sequence of tasks. We are excited to introduce the community to CORA, and hope CORA can aid in the development, testing, and understanding of new methods in the field of continual RL.

**Limitations** We study task sequences that share a high-dimensional observation space (images) and have a discrete action space. These RL tasks are finite-horizon, with episodic resets. In this work, we also primarily study video-game environments, with procedurally-generated variation. These assumptions are shaped by our perspective of the continual RL problem and current state of the field. We acknowledge that different points of view exist, backed by design choices which may differ from the conditions we study. For instance, we consider task cycling in this work, which may favor replay-based methods such as CLEAR that can retain data from all tasks in the sequence, after the first cycle. This protocol does not apply to a pure online learning setup, where no assumptions may be made on the structure and similarities of incoming data. We intend to relax these assumptions as new methods are developed and CORA evolves.

## ACKNOWLEDGEMENTS

We thank Jonathan Schwarz and David Rolnick for discussion on implementing baselines, and helpful feedback on this project. We additionally thank Abhinav Shrivastava with help running experiments. This work was supported by ONR MURI, ONR Young Investigator Program, and DARPA MCS.

## REFERENCES

- Dhruv Batra, Angel X Chang, Sonia Chernova, Andrew J Davison, Jia Deng, Vladlen Koltun, Sergey Levine, Jitendra Malik, Igor Mordatch, Roozbeh Mottaghi, et al. Rearrangement: A challenge for embodied ai. *arXiv preprint arXiv:2011.01975*, 2020.
- Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.
- 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 (JAIR)*, 47:253–279, 2013.
- Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.
- Simon Brodeur, Ethan Perez, Ankesh Anand, Florian Golemo, Luca Celotti, Florian Strub, Jean Rouat, Hugo Larochelle, and Aaron Courville. Home: A household multimodal environment. *arXiv preprint arXiv:1711.11017*, 2017.
- Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 532–547, 2018.
- Xinlei Chen, Abhinav Shrivastava, and Abhinav Gupta. Neil: Extracting visual knowledge from web data. In *Proceedings of (ICCV) International Conference on Computer Vision*, pp. 1409 – 1416, December 2013.
- Maxime Chevalier-Boisvert. gym-miniworld environment for openai gym. <https://github.com/maximecb/gym-miniworld>, 2018.
- Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. <https://github.com/maximecb/gym-minigrid>, 2018.
- Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pp. 2048–2056. PMLR, 2020.
- Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2013.
- Natalia Díaz-Rodríguez, Vincenzo Lomonaco, David Filliat, and Davide Maltoni. Don’t forget, there is more than forgetting: new metrics for continual learning, 2018.
- Kiana Ehsani, Winson Han, Alvaro Herrasti, Eli VanderBilt, Luca Weihs, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Manipulathor: A framework for visual object manipulation. In *CVPR*, 2021.
- Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pp. 1407–1416. PMLR, 2018.
- Jesse Farnbrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.
- Chuang Gan, Jeremy Schwartz, Seth Alter, Damian Mrowca, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, Megumi Sano, Kuno Kim, Elias Wang, Michael Lingelbach, Aidan Curtis, Kevin Tyler Feigl, Daniel Bear, Dan Gutfreund, David Daniel Cox, Antonio Torralba, James J. DiCarlo, Joshua B. Tenenbaum, Josh McDermott, and Daniel LK Yamins. ThreeDWorld: A platform for interactive multi-modal physical simulation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

- Xiaofeng Gao, Ran Gong, Tianmin Shu, Xu Xie, Shu Wang, and Song-Chun Zhu. Vrkitchen: an interactive 3d virtual environment for task-oriented learning. *arXiv preprint arXiv:1903.05757*, 2019.
- Dibya Ghosh, Jad Rahme, Aviral Kumar, Amy Zhang, Ryan P Adams, and Sergey Levine. Why generalization in rl is difficult: Epistemic pomdps and implicit partial observability. *Advances in Neural Information Processing Systems*, 34, 2021.
- Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *Conference on Robot Learning*, pp. 1025–1037. PMLR, 2020.
- William H Guss, Cayden Codel, Katja Hofmann, Brandon Houghton, Noboru Kuno, Stephanie Milani, Sharada Mohanty, Diego Perez Liebana, Ruslan Salakhutdinov, Nicholay Topin, et al. The minerl 2019 competition on sample efficient reinforcement learning using human priors. *arXiv preprint arXiv:1904.10079*, 2019.
- Assaf Hallak, Dotan Di Castro, and Shie Mannor. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.
- Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *arXiv preprint arXiv:1810.12488*, 2018.
- Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning. In *International Conference on Learning Representations*, 2021.
- David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.
- Khurram Javed and Martha White. Meta-learning representations for continual learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1820–1830, 2019.
- Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *International Conference on Machine Learning*, pp. 4940–4950. PMLR, 2021.
- Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, Jonathan Harper, Ervin Teng, Hunter Henry, Adam Crespi, Julian Togelius, and Danny Lange. Obstacle tower: A generalization challenge in vision, control, and planning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019.
- Harini Kannan, Danijar Hafner, Chelsea Finn, and Dumitru Erhan. Robodesk: A multi-task reinforcement learning benchmark. <https://github.com/google-research/robodesk>, 2021.
- Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–8. IEEE, 2016.
- Khimya Khetarpal, Shagun Sodhani, Sarath Chandar, and Doina Precup. Environments for lifelong reinforcement learning. *arXiv preprint arXiv:1811.10732*, 2018.
- Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *arXiv preprint arXiv:2012.13490*, 2020.
- Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*, 2021.
- James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences of the United States of America*, 114(13):3521–3526, 2017.
- Jens Kober and Jan Peters. Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, pp. 579–610. Springer, 2012.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017.

Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148, 2013.

Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. <https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail>, 2018.

Heinrich Küttler, Nantas Nardelli, Thibaut Lavril, Marco Selvatici, Viswanath Sivakumar, Tim Rocktäschel, and Edward Grefenstette. TorchBeast: A PyTorch Platform for Distributed RL. *arXiv preprint arXiv:1910.03552*, 2019. URL <https://github.com/facebookresearch/torchbeast>.

Heinrich Küttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Youngwoon Lee, Edward S Hu, and Joseph J Lim. IKEA furniture assembly environment for long-horizon complex manipulation tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021. URL <https://clvrai.com/furniture>.

Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information Fusion*, 58:52–68, 2020.

Vincenzo Lomonaco, Karan Desai, Eugenio Culurciello, and Davide Maltoni. Continual reinforcement learning in 3D non-stationary environments. In *Proceedings of the CVPR Workshop on Continual Learning in Computer Vision*, pp. 248–249, 2020.

Vincenzo Lomonaco, Lorenzo Pellegrini, Andrea Cossu, Antonio Carta, Gabriele Graffieti, Tyler L Hayes, Matthias De Lange, Marc Masana, Jary Pomponi, Gido M Van de Ven, et al. Avalanche: an end-to-end library for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3600–3610, 2021.

David Lopez-Paz and Marc’ Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 6467–6476, 2017.

Nicolò Lucchesi, Antonio Carta, and Vincenzo Lomonaco. Avalanche rl: a continual reinforcement learning library. *arXiv preprint arXiv:2202.13657*, 2022.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Zheda Mai, Ruiwen Li, Jihwan Jeong, David Quispe, Hyunwoo Kim, and Scott Sanner. Online continual learning in image classification: An empirical survey. *arXiv preprint arXiv:2101.10423*, 2021.

Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search of static linear policies is competitive for reinforcement learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 1805–1814, 2018.

Vegard Mella, Eric Hambro, Danielle Rothermel, and Heinrich Küttler. moolib: A Platform for Distributed RL. 2022. URL <https://github.com/facebookresearch/moolib>.

Martial Mermilliod, Aurélia Bugaiska, and Patrick BONIN. The stability-plasticity dilemma: investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in Psychology*, 4:504, 2013. ISSN 1664-1078. doi: 10.3389/fpsyg.2013.00504.

T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*, 2015.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Martin Mundt, Yong Won Hong, Iuliia Plushch, and Visvanathan Ramesh. A wholistic view of continual learning with deep neural networks: Forgotten lessons and the bridge to active and open world learning. *arXiv preprint arXiv:2009.01797*, 2020.

Hadi Nekoei, Akilesh Badrinaaraayanan, Aaron Courville, and Sarath Chandar. Continuous coordination as a realistic scenario for lifelong learning. In *International Conference on Machine Learning*, pp. 8016–8024. PMLR, 2021.

Fabrice Normandin, Florian Golemo, Oleksiy Ostapenko, Pau Rodriguez, Matthew D Riemer, Julio Hurtado, Khimya Khetarpal, Dominic Zhao, Ryan Lindeborg, Thimothée Lesort, et al. Sequoia: A software framework to unify continual learning research. *arXiv preprint arXiv:2108.01005*, 2021.

OpenAI. Robogym. <https://github.com/openai/robogym>, 2020.

Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvári, Satinder Singh, Benjamin Van Roy, Richard S. Sutton, David Silver, and Hado van Hasselt. Behaviour suite for reinforcement learning. *CoRR*, abs/1908.03568, 2019.

German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

Simone Parisi, Hany Abdulsamad, Alexandros Paraschos, Christian Daniel, and Jan Peters. Reinforcement learning vs human programming in tetherball robot games. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2015.

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *CoRR*, abs/1802.09464, 2018.

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

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8494–8502, 2018.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

Mark B Ring. CHILD: A first step towards continual learning. In *Learning to learn*, pp. 261–292. Springer, 1998.

Mark Bishop Ring. *Continual learning in reinforcement environments*. PhD thesis, University of Texas at Austin Austin, Texas 78712, 1994.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In *Advances in Neural Information Processing Systems*, volume 32, pp. 350–360, 2019.

Paul Ruvolo and Eric Eaton. ELLA: An efficient lifelong learning algorithm. In *Proceedings of the International Conference on Machine learning (ICML)*, pp. 507–515, 2013.

Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Kuttler, Edward Grefenstette, and Tim Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

Manolis Savva, Angel X Chang, Alexey Dosovitskiy, Thomas Funkhouser, and Vladlen Koltun. Minos: Multimodal indoor simulator for navigation in complex environments. *arXiv preprint arXiv:1712.03931*, 2017.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9339–9347, 2019.

- Jonathan Schwarz, Daniel Altman, Andrew Dudzik, Oriol Vinyals, Yee Whye Teh, and Razvan Pascanu. Towards a natural benchmark for continual learning. In *Proceedings of the NeurIPS Workshop on Continual Learning*, 2018a.
- Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pp. 4528–4537, 2018b.
- Bokui Shen, Fei Xia, Chengshu Li, Roberto Martín-Martín, Linxi Fan, Guanzhi Wang, Shyamal Buch, Claudia D’Arpino, Sanjana Srivastava, Lyne P Tchapmi, et al. igibson, a simulation environment for interactive tasks in large realistic scenes. *arXiv preprint arXiv:2012.02924*, 2020.
- Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10740–10749, 2020.
- David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.
- Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, et al. Habitat 2.0: Training home assistants to rearrange their habitat. *arXiv preprint arXiv:2106.14405*, 2021.
- Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.
- S. Thrun and Tom Michael Mitchell. Lifelong robot learning. *Robotics Auton. Syst.*, 15:25–46, 1995.
- Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.
- Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David and Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, pp. 1–5, 2019.
- Jianren Wang, Xin Wang, Yue Shang-Guan, and Abhinav Gupta. Wanderlust: Online continual object detection in the real world. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10829–10838, 2021.
- Jiayi Weng, Min Lin, Shengyi Huang, Bo Liu, Denys Makoviichuk, Viktor Makoviychuk, Zichen Liu, Yufan Song, Ting Luo, Yukun Jiang, et al. Envpool: A highly parallel reinforcement learning environment execution engine. *arXiv preprint arXiv:2206.10558*, 2022.
- Maciej Wołczyk, Michał Zając, Razvan Pascanu, Łukasz Kuciński, and Piotr Miłoś. Continual world: A robotic benchmark for continual reinforcement learning. *arXiv preprint arXiv:2105.10919*, 2021.
- Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11097–11107, 2020.
- Eliot Xing, Abhinav Gupta, Sam Powers, and Victoria Dean. Kitchenshift: Evaluating zero-shot generalization of imitation-based policy learning under domain shifts. In *NeurIPS 2021 Workshop on Distribution Shifts: Connecting Methods and Applications*, 2021.
- Claudia Yan, Dipendra Misra, Andrew Bennett, Aaron Walsman, Yonatan Bisk, and Yoav Artzi. Chalet: Cornell house agent learning environment. *arXiv preprint arXiv:1801.07357*, 2018.
- Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pp. 1094–1100. PMLR, 2020.

Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018.

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3357–3364. IEEE, 2017.

Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

## A EXTENDED RELATED WORK

**Environments and tasks** Historically, RL agents were evaluated on simple control tasks with state-based inputs from the OpenAI Gym (Brockman et al., 2016) or DeepMind Control Suite (Tassa et al., 2018). Some of these tasks have been shown to be easily solvable by random search algorithms (Mania et al., 2018) and thus should not be considered as sufficiently difficult for comparing algorithms. Leveraging physics simulators, many environments have been proposed that involve robots, fixed in place, for object manipulation tasks of varying complexity (Plappert et al., 2018; Rajeswaran et al., 2018; Gupta et al., 2020; Xing et al., 2021; OpenAI, 2020; Kannan et al., 2021; James et al., 2020; Zhu et al., 2020; Yu et al., 2020; Lee et al., 2021). Learning policies for robot manipulation is challenging, compounded by the exploration difficulty of the task, continuous action spaces, and the sample inefficiency of RL algorithms.

In this work, we choose environments with discrete action spaces, in order to use a single output policy across multiple tasks, rather than use separate output heads for different tasks. We make this decision considering the capabilities of current methods. For environments with continuous action spaces, task-agnostic single-headed architectures may be infeasible currently, as in (Wołczyk et al., 2021), so we leave this for future work. Furthermore, it is beneficial for continual RL for tasks to share a consistent observation space, allowing for the creation of common policies that can leverage task similarities. If environments rely on state-based inputs such as the positions of objects, it is usually the case that the state space changes for different tasks. To ensure a consistent observation space, we pick environments which offer image-based observation spaces.

RL is also frequently evaluated on video game-like environments, most commonly Atari (Bellemare et al., 2013), among others (Chevalier-Boisvert et al., 2018; Chevalier-Boisvert, 2018; Vinyals et al., 2017; Beattie et al., 2016; Kempka et al., 2016; Juliani et al., 2019; Guss et al., 2019; Cobbe et al., 2020; Küttler et al., 2020). In this work, we reproduce prior continual RL results on Atari (Schwarz et al., 2018b; Rolnick et al., 2019). We also define new task sequences using Procgen (Cobbe et al., 2020) and MiniHack (Samvelyan et al., 2021). MiniHack is designed to isolate more tractable subproblems in the highly challenging NetHack (Küttler et al., 2020) environment. We believe Procgen and MiniHack, with fast, procedurally-generated, stochastic environments to be better testbeds for continual RL onwards.

Beyond video game environments, many home environment simulators have been proposed recently (Brodeur et al., 2017; Savva et al., 2017; Puig et al., 2018; Yan et al., 2018; Gao et al., 2019; Savva et al., 2019) which offer visually-realistic scenes for evaluating Embodied AI. Among these types of environments, we highlight AI2-THOR (Kolve et al., 2017), Habitat 2.0 (Szot et al., 2021), iGibson (Shen et al., 2020), Sapien (Xiang et al., 2020), and ThreeDWorld (Gan et al., 2021), which feature a wide range of household objects and scene-level interaction tasks. In particular, AI2-THOR, Habitat 2.0, and iGibson provide multiple home scenes based on real-world data. These different scenes are useful for applying realistic domain shift between tasks and evaluating forward transfer when learning later tasks. We choose AI2-THOR as a simulation environment in this benchmark because it offers a higher-level discrete action space, compared to Habitat 2.0 or iGibson at the time of development, along with a diverse set of demonstrations released in ALFRED (Shridhar et al., 2020). We also note that recent work using AI2-THOR has done evaluations with object manipulation tasks (Shridhar et al., 2020; Batra et al., 2020; Ehsani et al., 2021), paving the way for more complex action spaces.

## B BACKGROUND

Formally, we consider each task  $\mathcal{T}$  as a finite, discrete-time Markov decision process (MDP), represented by a tuple  $\langle S, A, T, r, \rho_0 \rangle$ , with state space  $S$ , action space  $A$ , state transition probability function  $T$ , reward function  $r$ , and probability distribution  $\rho_0$  on the initial states  $S_0 \subset S$ . In the standard reinforcement learning setup, the goal is to learn a policy  $\pi(a|s)$  which maximizes the expected return  $\mathcal{R}$ , where the return of a state  $s$  is defined as the sum of (discounted) rewards over a finite-length episode from state  $s$ .

We refer the reader to (Zhang et al., 2018; Ghosh et al., 2021; Kirk et al., 2021), on which we base our discussion of generalization in RL using a Contextual MDP (CDMP) (Hallak et al., 2015). A state  $s \in S$  can be decomposed as  $(c, s') \in S_C$ , where  $s' \in S'$  is the underlying state and  $c \in C$  is the context, such that  $S = C \times S'$ . We assume that the context is not observed by the agent, so the CDMP is a partially observable MDP (POMDP) with observation space  $O$ .

The context  $c$  remains fixed throughout an episode, and determines the environment variation. The initial state distribution may be factorized as  $\rho_0(s) := p(c)\rho_0(s'|c)$ , where  $p(c)$  is called the context distribution, used to determine collections of training and testing environments. Formally, we consider context sets  $C_{train}$  and  $C_{test}$ , where the policy is trained on training context-set CMDP  $\mathcal{T}|_{C_{train}}$  and evaluated on the testing context-set CMDP  $\mathcal{T}|_{C_{test}}$ .

We denote the expected return in the CMDP as  $\mathbb{R}(\pi, \mathcal{T}) := \mathbb{E}_{c \sim p(c)}[\mathcal{R}(\pi, \mathcal{T}|_c)]$ . The objective for the policy  $\pi$  is to maximize the expected return on the testing context set,  $\mathbf{R}(\pi, \mathcal{T}|_{C_{test}})$ . Furthermore, the generalization gap between train and test performance can be measured by  $\mathbf{R}(\pi, \mathcal{T}|_{C_{train}}) - \mathbf{R}(\pi, \mathcal{T}|_{C_{test}})$ .

In Progen, we consider *in-distribution* generalization on unseen environment contexts, where  $C_{train}$  and  $C_{test}$  are disjoint sets composed of i.i.d samples of  $C$ . In particular,  $c$  is a random seed which determines how the game level procedurally generates. For the easy difficulty setting of Progen,  $C_{train}$  is composed of 200 fixed seeds, while  $C_{test} = C$  is uniform over all seeds.

In MiniHack, we consider *out-of-distribution* generalization, namely extrapolation along different environment factors. In addition to a random seed, any MiniHack environment instance is also determined by its `des-file`, which controls map layout as well as placement of environment features, monsters, and objects. For example, the MiniHack task Corridor-R5 has one `des-file` associated with it, while KeyRoom-S5 has defined its own separate one. Thus, each (train, test) task pair tests extrapolation along environment variations such as room size, number of rooms, obstacles, or lighting. We refer the reader to Appendix C of the MiniHack paper for further details on variation (Samvelyan et al., 2021), and Appendix C.3 of our paper for the full list of MiniHack task pairs we use. Similarly, CHORES Gen-MultiTraj evaluates out-of-distribution generalization on unseen factors such as room scene and object of interest.

For a continual reinforcement learning setup, we further consider a sequence of  $N$  tasks,  $\mathcal{S}_N := (\mathcal{T}_0 \dots \mathcal{T}_{N-1})$ . The agent trains on task  $\mathcal{T}_i$  at timesteps in the interval  $[A_i, B_i]$ , where  $A_i$  and  $B_i$  are the task boundaries denoting the start and end, respectively, of task  $\mathcal{T}_i$ . We cycle through the tasks  $M$  times, so the full task sequence  $\mathcal{S}_{NM}$  has length  $N \cdot M$ . We assume that the tasks are drawn from some world collection  $\mathcal{W}$ , where the dimensions of the observation space and action space are consistent across all tasks from  $\mathcal{W}$ . This enables us to train one model, with a single output policy layer, over the task sequence. In this work, we consider  $\mathcal{W}$  to be Atari games, Progen games, NetHack dungeons for MiniHack, or AI2-THOR scenes for CHORES. We further assume that tasks in  $\mathcal{W}$  have some shared structure, for instance related to dynamics (ie. enemies hurt players) or rewards (ie. survive longer), which humans would also learn to exploit in order to perform all tasks in  $\mathcal{W}$ .

There are a couple ways to view the benefits of defining the learning problem in this sequential manner, compared to training an expert for each task or training via multi-task learning. The first is through the lens of a robotic agent, operating in the real world in a way similar to humans. Such an agent should learn and adapt to new settings as they are encountered, without forgetting prior learned behavior. Training multiple tasks in parallel or training on each task individually with human-specified boundaries makes additional assumptions, which encourages the development of methods that are ill-suited to the needs of lifelong robotic agents in the real-world.

The second way to view the benefits of continual RL is that sequentially learning tasks is a simple and effective way to induce a *non-stationary* learning process. Any component of the MDP may change on task switch, and the agent should be capable of handling such distribution shifts. While our benchmarks are not imbued with all the ways in which the real world may change, we see this way of modeling the learning problem as a step towards the final goal of real-world embodied agents, for future work to build off.

## C CORA DETAILS

All code, including hyperparameters, is available here: [https://github.com/AGI-Labs/continual\\_rl](https://github.com/AGI-Labs/continual_rl).

We divide additional details for CORA in this section into: design objectives for CHORES (Appendix C.1), details of each CHORES task sequence (Appendix C.2), list of MiniHack task pairing (Appendix C.3), examples of initial observations for video-game task sequences (Appendix C.4), experiment runtimes (Appendix C.7), baseline implementation details (Appendix C.5), hyperparameters (Appendix C.6), Atari experiment results (Appendix C.8), additional Procgen figures (Appendix C.9), final performance tables on all environments (Appendix C.10), and avenues for future work (Appendix C.11).

### C.1 CHORES DESIGN OBJECTIVES

	Difficulty	Test Type	Num traj. per task	Scene	Task	Object
Mem-VaryScene	easier	memorization	1	Δ, bath	put in bathtub	hand towel
Mem-VaryTask	easier	memorization	1	bath, Room 402	Δ	toilet paper (TP)
Mem-VaryObject	easier	memorization	1	kitchen, Room 24	clean object	Δ
Gen-MultiTraj	harder	generalization	3	Δ, kitchen	Cool & put in sink	Δ

	Task A	Task B	Task C
Mem-VaryScene	Room 402 ( $r = 12$ )	Room 419 ( $r = 12$ )	Room 423 ( $r = 12$ )
Mem-VaryTask	hang TP ( $r = 12$ )	put 2 TP in cabinet ( $r = 24$ )	put 2 TP on counter ( $r = 24$ )
Mem-VaryObject	fork ( $r = 18$ )	knife ( $r = 18$ )	spoon ( $r = 18$ )
Gen-MultiTraj	Room 19, cup ( $r = 18$ )	Room 13, sliced potato ( $r = 31$ )	Room 2, sliced lettuce ( $r = 31$ )

Table 2: Summary of the four CHORES benchmarks. The first three are memorization tasks, and are evaluated on the training environment. The fourth is a harder generalization task, with 3 trajectories per task to initialize the scene and task parameters. We also summarize which scene each task is in, what task it performs, and what objects it utilizes. We categorize each CHORES by what the task sequences varies. The  $r$  values in parentheses show the minimum return for solving the task.

**Goal communication** All CORA benchmarks other than CHORES use video game environments, where the visual differences between the tasks may have been sufficient for the agent to know what they are supposed to do, in order to receive reward. For instance in Atari, 0-SpaceInvaders is distinct enough in appearance from 2-BeamRider that no further task specification is required, see Appendix C.4 Figure 6. However, since all CHORES take place in a fixed set of rooms, the observation that the agent receives on its own is insufficient to distinguish task boundaries with. In this work, we use subgoal images in CHORES to communicate task intentions to the agent. In real-world settings, this could be achieved by a human demonstrating a task and taking pictures at critical points during the task to give to a robotic agent. Future work may leverage the language annotations ALFRED provides with each demonstration trajectory for alternate as more convenient forms of communication would be useful for robotic agents to employ.

**Task constraints** To make the benchmark as accessible for the community, our aim was for each task used by CHORES to be individually solvable in under five hours using a machine with 16 vCPUs, 64 GB of RAM, and a Titan X GPU. Given the nature of simulating realistic environments, this corresponds to a budget of around 1 million frames per task. Additionally, since continual RL ultimately should be deployed onto robotic agents in the real world, modest sample budgets align with what will likely be feasible with real world learning.

Most existing policies may not be sample efficient enough to learn complex tasks in this amount of time. However, by providing sequences of simple tasks that are at the edge of what is currently achievable, we hope to move beyond this boundary and encourage the development of algorithms that are successful under these conditions. We also provide one complex task as an example of what is possible moving forward and for what we hope will be achievable in future CHORES benchmarking.

**Task selection** The CHORES tasks were (by necessity) somewhat more hand-picked. These were selected in the following way:

1. We used ALFRED to generate a new set of trajectories for the latest AI2-THOR version (needed for headless rendering to use on our cluster) using ALFRED’s defined set of tasks.
2. Based on our defined axes of variation (e.g. varying objects), we filtered successfully generated tasks into clusters that met our criteria.

3. From this filtered set, we selected tasks to maximize diversity (e.g. more than just pick-and-place).

The selection process was done more out of necessity than the ideal, but we believe the tasks cover the desired goals of the benchmark more than adequately.

## C.2 CHORES DETAILS

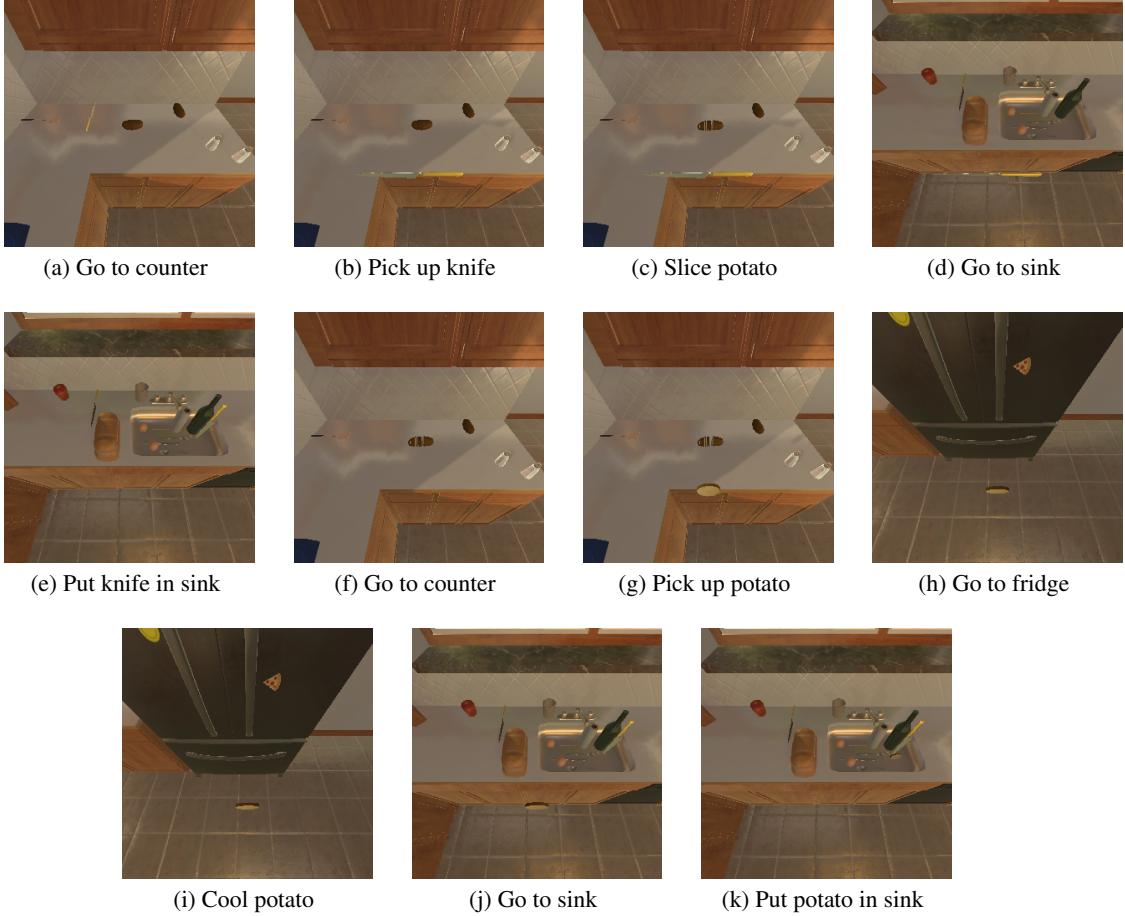


Figure 5: Visualization of one ALFRED trajectory used to define a task in CHORES.

To start, we propose three memorization-based CHORES, each of which tests an agent’s robustness to a particular type of domain shift. Each task within these three CHORES is intended to be relatively easy, using only one trajectory to set the environment parameters and evaluating in the same scene as during training. We additionally propose one harder CHORES to evaluate generalization. This last CHORES has more complex tasks, with a set of three trajectories per task to initialize the environment from, and evaluation is also done in unseen settings from a different 3 trajectories. These benchmarks are summarized in Table 2 in the Appendix. In all cases, the locations of moveable, interactable objects are randomized between trajectories.

The first task sequence, which we refer to as **Mem-VaryRoom**, keeps the task type and task object the same, while changing the room scene the agent interacts in to different bathrooms. The agent is trained to find a hand towel and place it in the bath tub of Room 402 for 1M steps, then in Room 419, then in Room 423. We then cycle through the environments again, to evaluate how much faster learning each environment is the second time.

The second task sequence, **Mem-VaryTask**, follows the same pattern but holds the current room and object constant, while changing the task. The agent is trained in the same bathroom to change a roll of toilet paper on a hanger, then to put two rolls of toilet paper in the cabinet, and finally to place two rolls on the countertop.

The third task sequence, **Mem-VaryObject**, holds the current room and task constant but changes the object. In kitchen 24 the agent is tasked to clean a fork, then clean a knife, then clean a spoon. Cleaning is done by putting an object under running water from a faucet. For the first two tasks, after cleaning the agent must put the object on the counter top, and in the third it must put it in the cabinet.

The fourth task sequence, **Gen-MultiTraj**, uses a task where an agent takes an object, puts it in the fridge to cool it, removes it, and then places it in the sink. With this base task, the task sequence is as follows: (a) in kitchen 19, the agent performs the task with a cup; (b) in kitchen 13, the agent must slice a potato, then perform the task with the sliced potato; (c) in kitchen 2, the agent must slice lettuce, then perform the task with the sliced lettuce. The key difference from the previous task sequences is that each task in the fourth CHORES is evaluated on unseen settings initialized from three possible heldout demonstrations trajectories, testing an agent’s ability to generalize.

In Figure 5, we visualize all subgoal images for one trajectory of the Gen-MultiTraj potato task (task 2).

**Reward details** Unlike ALFRED which reports the number of subgoals achieved, we report the episode returns for consistency with the other benchmarks, clipped to a minimum value of -10. Extremely negative values occur when the agent performs a particularly suboptimal action for the duration of the episode, until the maximum step limit of 1000 is hit. Without clipping, this occasional negative behavior completely drowns out the agent’s successes, both in visualization and metrics.

### C.3 MINIHACK TASK SEQUENCE

The MiniHack (train, test) paired task sequence we use is:

0	(Room-Random-5x5, Room-Random-15x15)
1	(Room-Dark-5x5, Room-Dark-15x15)
2	(Room-Monster-5x5, Room-Monster-15x15)
3	(Room-Trap-5x5, Room-Trap-15x15)
4	(Room-Ultimate-5x5, Room-Ultimate-15x15)
5	(Corridor-R2, Corridor-R5)
6	(Corridor-R3, Corridor-R5)
7	(KeyRoom-S5, KeyRoom-S15)
8	(KeyRoom-Dark-S5, KeyRoom-Dark-S15)
9	(River-Narrow, River)
10	(River-Monster, River-MonsterLava)
11	(River-Lava, River-MonsterLava)
12	(HideNSeek, HideNSeek-Big)
13	(HideNSeek-Lava, HideNSeek-Big)
14	(CorridorBattle, CorridorBattle-Dark)

This task sequence defines 15 pairs of (train, eval) environments and uses a total of 27 different environments. Some of the evaluation environments are used multiple times because those test environments have more than one related train environment. In particular, this impacts task pairs 5 and 6; 10 and 11; 12 and 13.

### C.4 EXAMPLES FOR INITIAL OBSERVATIONS OF THE VIDEO GAME BENCHMARKS

We show examples of initial observations that an agent may get for each task in the Atari (Figure 6), Procgen (Figure 7), and MiniHack (Figure 8) task sequences. Note that visually, it is easy to distinguish the different tasks for Atari and Procgen. However, since MiniHack tasks use the same visual assets, it is more challenging to tell tasks apart from each other. This makes task boundary identification more difficult for algorithms. Furthermore, this also supports why CHORES provides agents with a goal image specifying the task to perform.

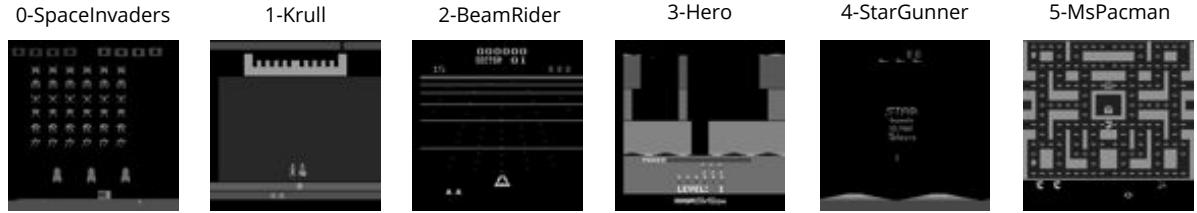


Figure 6: Examples of initial observations for each task in the 6 task Atari sequence.

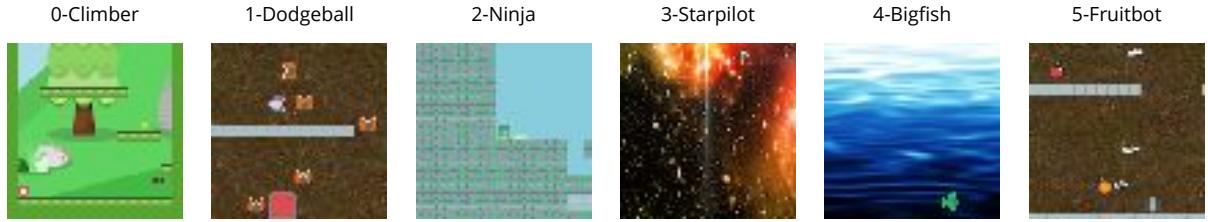


Figure 7: Examples of initial observations for each task in the 6 task Progen sequence

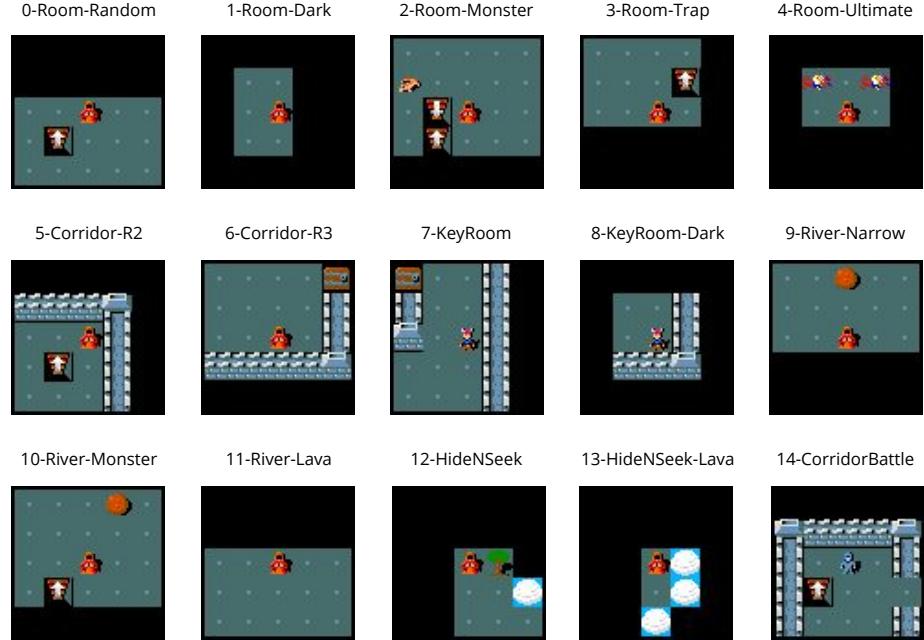


Figure 8: Examples of initial observations for each task in the 15 task MiniHack sequence. Observations are shown for the training task of each pair.

### C.5 BASELINE IMPLEMENTATION DETAILS

After discussion with the authors, the original P&C (Schwarz et al., 2018b) results collect the task’s budgeted number of samples from the environment for each of their progress and compress phases. Instead, we opt to collect half for each, rather than effectively doubling the budget for environment steps compared to other methods. This may be why we

observe P&C underperforming compared to the original paper’s results. Note that P&C evaluates using the knowledge base model, which is only updated in the second half of training on each task. As such, for the first half of each task (while the active column model is being updated), essentially a flat performance curve is reported for P&C.

Additionally, as stated in Appendix C.8, the original authors used a pre-release version of IMPALA, while we use the TorchBeast implementation of IMPALA. For CLEAR, we used a 25M frame replay buffer while the original CLEAR paper used a replay buffer of half the number of frames that the agent trains on. For two cycles, this would correspond to 300M frames, while the authors of CLEAR used a replay buffer of 750M frames for their Atari experiment running 5 learning cycles. Due to system constraints, we are unable to run with the larger replay buffer size. Even with this difference, our implementation of CLEAR outperforms the results reported in the original paper. CLEAR uses reservoir sampling (Isele & Cosgun, 2018) to maintain a buffer that stores a uniform sample of all past experience.

We use a *single* output head across the task sequence, rather than separate heads for each task such as in (Wołczyk et al., 2021). We evaluate using the eval mode of policies. For all IMPALA-based methods, eval mode takes the argmax action instead of a stochastic sampling. In the case of P&C, we evaluate using the output of the knowledge base instead of the active column. While we use existing methods in the manner in which they were designed, we would like to encourage future methods to have parity between train and evaluation policies.

**Network architecture** For Atari (Appendix C.8), we use the “shallow” model without an LSTM from the IMPALA paper (Espeholt et al., 2018). We also use this model for MiniHack (Section 6.2) and CHORES (Section 6.3). For Procgen (Section 6.1), we use the “deep” residual model without an LSTM from the IMPALA paper, following (Jiang et al., 2021; Cobbe et al., 2020). We include Procgen results from an earlier version of this paper that used the “shallow” model in Appendix C.9.

## C.6 HYPERPARAMETERS

Hyperparameter	Shared
Num. actors	64
Learner threads	2
Batch size	32
Unroll length	20
Grad clip	40
Reward clip	$[-1, 1]$
Normalize rewards	No
Baseline cost	0.5
Entropy cost	0.01
Discount factor	0.99
LSTM	No
Network arch.	Nature CNN
Learning rate	$4e-4$
Optimizer	RMSProp $\alpha = 0.99$ $\epsilon = 0.01$ $\mu = 0$

Hyperparameter	EWC	Online EWC	P&C	CLEAR
EWC $\lambda$	10000	175	3000	
EWC, min. task steps	2e5	2e5		
Fisher samples	100	100	100	
Normalize Fisher	No	Yes	Yes	
Online EWC $\gamma$		0.99	0.99	
KL cost			1.0	
Batch ratio (novel-replay)				50-50
Policy cloning cost				0.01
Value cloning cost				0.005
Replay buffer size				25e6

Table 3: Hyperparameters for baselines on the Atari. For Atari, the network architecture is the Nature-CNN model from the DQN paper (Mnih et al., 2015). For Procgen, we use the same values, except CLEAR’s replay buffer size is reduced to 5e6, and across all baselines, we switch the network architecture to the “deep” residual model from the original IMPALA paper (Espeholt et al., 2018).

Hyperparameter	Shared	Hyperparameter	Shared
Reward clip	No	Num. actors	10
Normalize rewards	Yes	Batch size	10
Entropy cost	0.001	Unroll length	80
Discount factor	0.999		
Learning rate	$2e-4$		
Optimizer	RMSProp $\epsilon = 1e-6$		

(a) MiniHack

(b) CHORES

Table 4: Hyperparameters for baselines on (a) MiniHack, following values used by [Samvelyan et al. \(2021\)](#); (b) CHORES. We report values changed from those used for Atari.

We report hyperparameters used by the baselines for Atari in Table 3. Note that the scale of the losses differ between EWC vs. Online EWC and P&C, see Appendix C.2 of the P&C paper ([Schwarz et al., 2018b](#)). For EWC, we conducted a hyperparameter search with learning rate from  $[1e-5, 1e-4, 4e-4, 6e-4]$  and EWC  $\lambda$  from  $[1, 100, 175, 500, 1000, 1500, 3000, 5000, 10000]$ . For Online EWC and P&C, we chose EWC  $\lambda$  from  $[175, 3000, 10000]$  and Online EWC  $\gamma$  from  $[0.95, 0.99, 0.999, 0.9999]$ . For CLEAR, we chose the replay buffer size from  $[5e6, 20e6, 25e6]$ .

For Procgen, we use the same hyperparameters as for Atari, except CLEAR’s replay buffer size is reduced to  $5e6$ , and across all baselines we switch the network architecture to the “deep” residual model from the original IMPALA paper ([Espeholt et al., 2018](#)). For MiniHack, we use hyperparameters from [Samvelyan et al. \(2021\)](#), which are shown in Table 4 (a). We also reduce CLEAR’s replay buffer size to  $10e6$ . For CHORES, we report the hyperparameters used in Table 4 (b).

## C.7 EXPERIMENT RUNTIMES

When we run using a single GPU on a machine with 128 GB of RAM, and 32 or 40 vCPUs, we observe the following runtime averages across the entire experiments:

- MiniHack: 45 hours (1.5 hr/task)
- Procgen: 54 hours (1.8 hr/task)
- Atari: 129 hours (10.8 hr/task) – which would have taken 323 hours to run the original 5-cycle experiment
- CHORES: discussed in Section C.1, around 5 hours per task to train.

Atari, as the standard baseline, is what we aimed to improve upon with the selection of Procgen and MiniHack, and indeed we see a speedup of around 6-7x. CHORES, which uses the visually realistic AI2THOR home simulation environment, is slower, but still 2x as fast as Atari per task.

We note that concurrent with our work, advances ([Weng et al., 2022](#); [Mella et al., 2022](#)) have been made in accelerating the Atari simulator (and other CPU-based environments) using C++ threads and by bypassing the Python GIL. This has lowered the compute requirements for running longer Atari experiments, which were previously only possible with industry-level compute. We are excited for the possibilities opened by these advances, for future work on continual RL incorporating longer task sequences, and the more difficult Atari settings introduced by ([Machado et al., 2018](#); [Farebrother et al., 2018](#)).

## C.8 ATARI RESULTS

We include the standard, proven Atari task sequence as a benchmark, in order to validate our baseline implementations on an existing standard. The reproduction of these Atari results were developed over hundreds of hours, including time spent analyzing papers for algorithm details, corresponding with the original authors, tuning hyperparameters, and running many seeds of Atari experiments, each of which takes hundreds of millions of frames. These results were reproduced using a university server cluster and several thousand dollars of AWS credits, compared to the industry-level compute that the original authors (from DeepMind) [Schwarz et al. \(2018b\)](#) and [Rolnick et al. \(2019\)](#) had access to. This is one of the primary reasons we are advocating for more compute-friendly continual RL benchmarks. It is also the reason that we were only able to run 2 learning cycles for these Atari results instead of the intended 5 cycles.

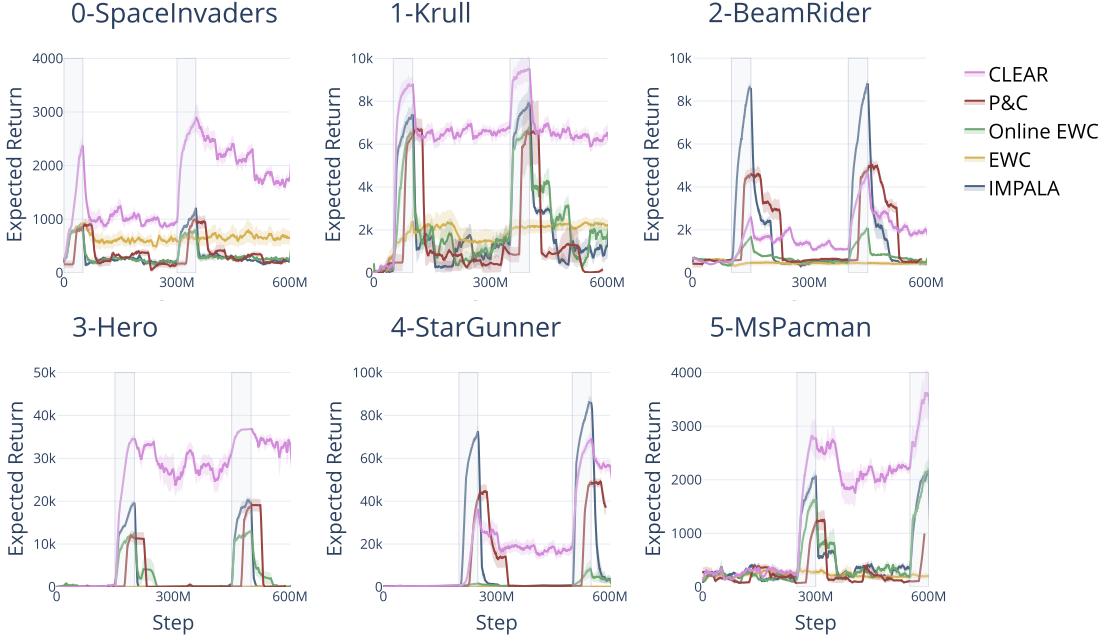


Figure 9: Results for Continual Evaluation ( $\mathcal{C}$ ) on the 6 Atari task sequence from (Rolnick et al., 2019; Schwarz et al., 2018b). Due to compute constraints, we only train for 2 cycles compared to the original experiments which used 5 learning cycles. IMPALA is the baseline learning algorithm that the other methods for continual RL build off. Gray shaded rectangles show when the agent trains on each task.

We use the full 18-dim action space for this task sequence. The observation space is (84, 84) grayscale images, and the agent receives a framestack of 4. The Atari games used are fully deterministic, and following the prior continual RL work on Atari, we do not apply sticky actions (Machado et al., 2018).

Atari results are shown in Figure 9, and we compare them against the results presented in Rolnick et al. (2019). Notably, on almost all Atari tasks, our implementations outperform the results reported in CLEAR. This may be because we use the TorchBeast (Küttler et al., 2019) implementation of IMPALA, while the results in Rolnick et al. (2019) and Schwarz et al. (2018b) use an earlier, pre-release version of IMPALA.

**Benchmark analysis:** Summary metrics are available in Table 1. From these, we observe that Atari does effectively test for robustness to catastrophic forgetting, but exhibits nearly no transfer. Looking at the diagnostic transfer in Table 12, we observe no transfer, likely because the six Atari tasks used are too distinct from each other.

**Algorithm design:** From the continual evaluation results in Figure 9, we can see that CLEAR outperforms the other baselines at both recall and plasticity on Atari, which matches the original results by (Rolnick et al., 2019). EWC maintains a flat return curve for early tasks, which is consistent, losing plasticity and failing to learn the later tasks. P&C largely maintains its plasticity, but we observe considerably more forgetting than was reported. We discuss this disparity in Appendix C.5.

### C.9 ADDITIONAL PROCGEN FIGURES

For readability, we provide an alternative plot (Figure 10) of the Progen results shown in Section 6.1, Figure 2, without the CLEAR baseline. Additionally, in Figure 11, we report baseline results on Progen using the “shallow” model. The updated Progen results in Section 6.1, Figure 2 use the “deep” residual model.

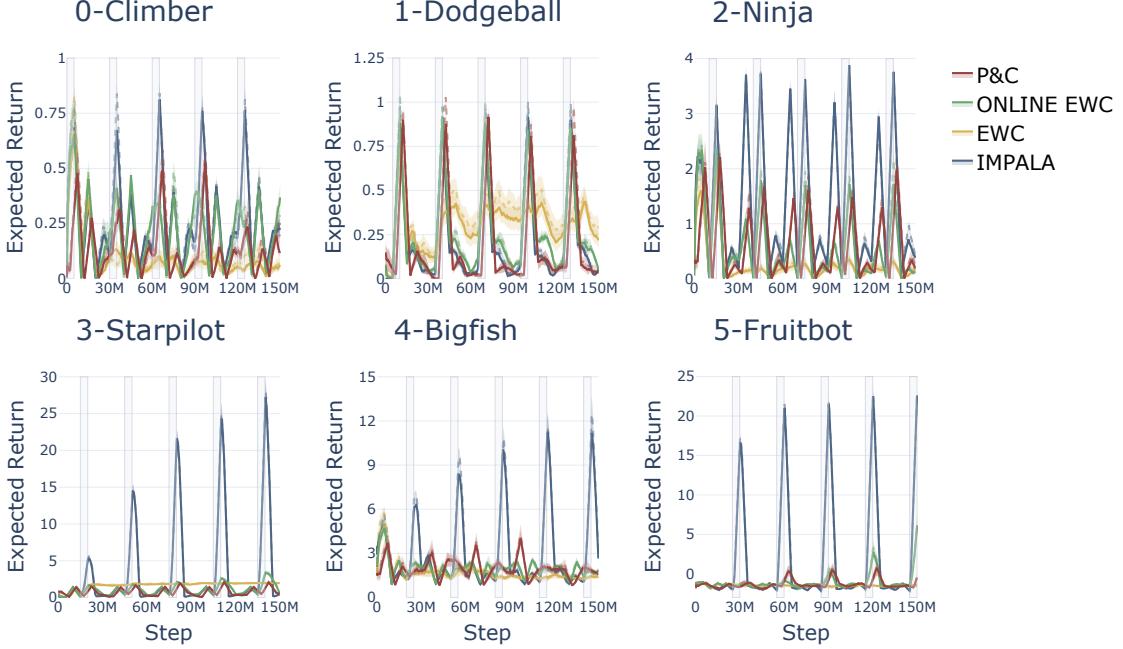


Figure 10: Results for Continual Evaluation ( $\mathcal{C}$ ) on the 6 Progen tasks. These are the same results as shown in Figure 2, but with CLEAR removed and rescaled to better visualize the baselines.

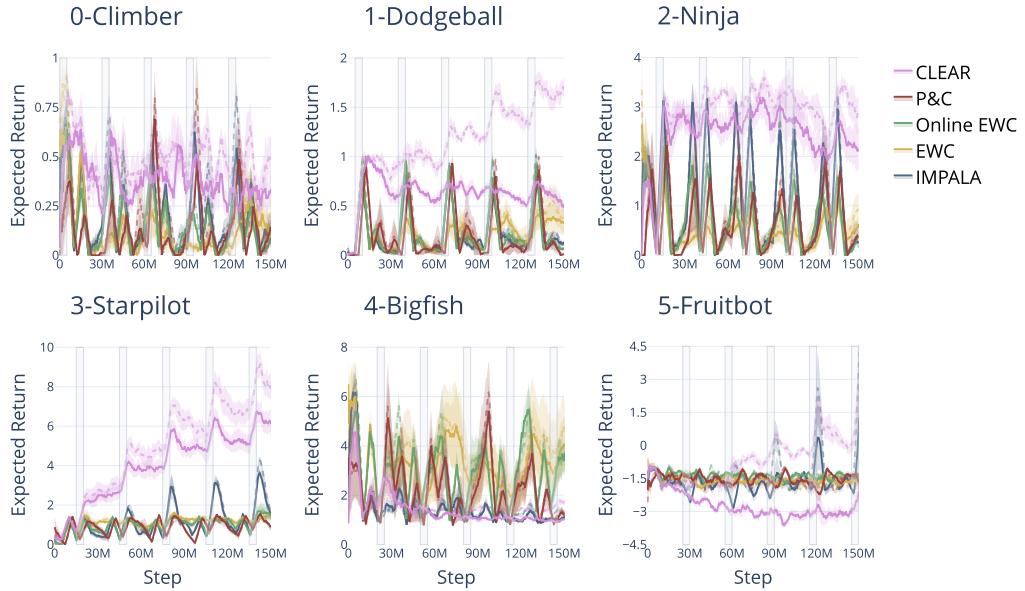


Figure 11: Results for Continual Evaluation ( $\mathcal{C}$ ) on the 6 Progen tasks, based on recommendations by Igl et al. (2021). The solid line shows evaluation on unseen testing environments; the dashed line shows evaluation on training environments. Gray shaded rectangles show when the agent trains on each task. These results use the “shallow” model from the IMPALA paper (Espeholt et al., 2018), while the results in Section 6.1, Figure 2 use the “deep” residual model.

## C.10 FINAL PERFORMANCE TABLES

### C.10.1 ATARI, FINAL PERFORMANCE TABLES

Task	CLEAR	P&C	Online EWC	EWC	IMPALA
0-SpaceInvaders	<b>1766.75 ± 89.36</b>	209.14 ± 55.91	240.19 ± 65.36	654.45 ± 134.16	247.64 ± 39.18
1-Krull	<b>6542.51 ± 410.25</b>	156.65 ± 141.62	1714.31 ± 532.34	2210.76 ± 290.60	1249.52 ± 592.68
2-BeamRider	<b>2002.95 ± 212.14</b>	627.60 ± 36.22	491.86 ± 68.31	458.69 ± 109.92	425.58 ± 108.48
3-Hero	<b>33604.29 ± 1487.71</b>	289.25 ± 289.25	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
4-StarGunner	<b>56365.93 ± 2881.83</b>	37515.49 ± 3692.96	3421.75 ± 1453.06	140.03 ± 87.05	3495.73 ± 1774.20
5-MsPacman	<b>3536.24 ± 400.45</b>	996.19 ± 58.19	2171.94 ± 229.90	201.91 ± 58.12	2104.36 ± 103.78

Table 5: Comparison of final performance (mean±SEM) between methods in the environments for each Atari task.

### C.10.2 PROCGEN, FINAL PERFORMANCE TABLES

Task	CLEAR	P&C	Online EWC	EWC	IMPALA
0-Climber	<b>0.74 ± 0.04</b>	0.12 ± 0.02	0.37 ± 0.06	0.06 ± 0.02	0.22 ± 0.04
1-Dodgeball	<b>1.35 ± 0.05</b>	0.04 ± 0.01	0.05 ± 0.01	0.22 ± 0.04	0.06 ± 0.01
2-Ninja	<b>3.89 ± 0.18</b>	0.20 ± 0.06	0.11 ± 0.04	0.20 ± 0.08	0.38 ± 0.06
3-Starpilot	<b>38.29 ± 1.45</b>	0.52 ± 0.09	0.53 ± 0.10	1.95 ± 0.06	0.44 ± 0.09
4-Bigfish	<b>11.31 ± 0.58</b>	1.63 ± 0.14	1.81 ± 0.18	1.39 ± 0.15	2.65 ± 0.18
5-Fruitbot	<b>25.81 ± 0.22</b>	-0.46 ± 0.42	6.16 ± 1.24	-1.44 ± 0.07	22.62 ± 0.33

Table 6: Comparison of final performance (mean±SEM) between methods in the *unseen testing* environments for each Procgen task.

Task	CLEAR	P&C	Online EWC	EWC	IMPALA
0-Climber	<b>0.81 ± 0.05</b>	0.19 ± 0.05	0.29 ± 0.05	0.07 ± 0.03	0.24 ± 0.04
1-Dodgeball	<b>2.19 ± 0.08</b>	0.05 ± 0.01	0.05 ± 0.01	0.29 ± 0.05	0.05 ± 0.01
2-Ninja	<b>3.92 ± 0.18</b>	0.27 ± 0.08	0.12 ± 0.03	0.20 ± 0.07	0.42 ± 0.09
3-Starpilot	<b>38.76 ± 1.52</b>	0.56 ± 0.10	0.52 ± 0.10	1.99 ± 0.06	0.43 ± 0.09
4-Bigfish	<b>13.90 ± 0.55</b>	1.52 ± 0.09	1.86 ± 0.18	1.60 ± 0.22	2.88 ± 0.27
5-Fruitbot	<b>27.04 ± 0.21</b>	-0.49 ± 0.38	6.56 ± 1.19	-1.64 ± 0.05	22.58 ± 0.41

Table 7: Comparison of final performance (mean±SEM) between methods in the *training* environments for each Procgen task.

## C.10.3 MINIHack, FINAL PERFORMANCE TABLES

Task	CLEAR	IMPALA
0-Room-Random	<b>0.24 ± 0.03</b>	-0.02 ± 0.08
1-Room-Dark	<b>0.33 ± 0.08</b>	-0.11 ± 0.07
2-Room-Monster	<b>0.30 ± 0.04</b>	-0.02 ± 0.08
3-Room-Trap	<b>0.29 ± 0.03</b>	0.03 ± 0.09
4-Room-Ultimate	<b>0.37 ± 0.07</b>	-0.09 ± 0.05
5-Corridor-R2	-0.84 ± 0.02	<b>-0.69 ± 0.07</b>
6-Corridor-R3	-0.85 ± 0.03	<b>-0.70 ± 0.07</b>
7-KeyRoom	<b>-0.32 ± 0.02</b>	-0.40 ± 0.00
8-KeyRoom-Dark	<b>-0.38 ± 0.00</b>	-0.40 ± 0.00
9-River-Narrow	<b>-0.18 ± 0.01</b>	-0.19 ± 0.04
10-River-Monster	-0.26 ± 0.01	<b>-0.20 ± 0.04</b>
11-River-Lava	-0.26 ± 0.01	<b>-0.20 ± 0.04</b>
12-HideNSeek	<b>-0.10 ± 0.03</b>	-0.17 ± 0.07
13-HideNSeek-Lava	<b>-0.09 ± 0.03</b>	-0.17 ± 0.07
14-CorridorBattle	<b>-0.33 ± 0.01</b>	<b>-0.33 ± 0.01</b>

Table 8: Comparison of final performance (mean±SEM) between methods in the *unseen testing* environments for each MiniHack task.

Task	CLEAR	IMPALA
0-Room-Random	<b>0.91 ± 0.02</b>	0.38 ± 0.12
1-Room-Dark	<b>0.67 ± 0.04</b>	0.23 ± 0.08
2-Room-Monster	<b>0.91 ± 0.01</b>	0.36 ± 0.12
3-Room-Trap	<b>0.92 ± 0.02</b>	0.38 ± 0.11
4-Room-Ultimate	<b>0.71 ± 0.03</b>	0.23 ± 0.07
5-Corridor-R2	<b>-0.14 ± 0.03</b>	-0.68 ± 0.06
6-Corridor-R3	-0.86 ± 0.02	<b>-0.68 ± 0.07</b>
7-KeyRoom	<b>-0.12 ± 0.02</b>	-0.20 ± 0.00
8-KeyRoom-Dark	<b>-0.18 ± 0.01</b>	-0.20 ± 0.00
9-River-Narrow	<b>0.05 ± 0.04</b>	-0.19 ± 0.04
10-River-Monster	-0.20 ± 0.02	<b>-0.17 ± 0.05</b>
11-River-Lava	-0.22 ± 0.01	<b>-0.18 ± 0.04</b>
12-HideNSeek	<b>0.17 ± 0.02</b>	0.03 ± 0.06
13-HideNSeek-Lava	<b>0.16 ± 0.02</b>	0.01 ± 0.06
14-CorridorBattle	<b>-0.18 ± 0.04</b>	-0.31 ± 0.02

Table 9: Comparison of final performance (mean±SEM) between methods in the *training* environments for each MiniHack task.

## C.10.4 CHORES, FINAL PERFORMANCE TABLE

Task	CLEAR	P&C	EWC
<b>Mem-VaryRoom</b>			
Room 402	$-5.27 \pm 4.73$	$0.00 \pm 0.00$	$2.73 \pm 1.47$
Room 419	$0.07 \pm 0.07$	$0.00 \pm 0.00$	$-6.53 \pm 3.47$
Room 423	$-3.33 \pm 3.33$	$0.00 \pm 0.00$	$-3.37 \pm 3.17$
<b>Mem-VaryTask</b>			
Hang TP	$-3.83 \pm 6.17$	$-6.50 \pm 3.25$	$-2.37 \pm 2.37$
Put TP on Counter	$-7.33 \pm 2.47$	$-6.53 \pm 3.27$	$-1.06 \pm 4.19$
Put TP in Cabinet	$12.71 \pm 9.15$	$-6.26 \pm 3.15$	$2.10 \pm 2.10$
<b>Mem-VaryObject</b>			
Clean Fork	$-2.06 \pm 7.53$	$-2.99 \pm 3.09$	$0.27 \pm 0.18$
Clean Knife	$-0.86 \pm 4.18$	$-3.13 \pm 3.03$	$3.67 \pm 1.73$
Clean Spoon	$1.36 \pm 6.71$	$-6.21 \pm 3.11$	$-2.97 \pm 3.27$
<b>Gen-MultiTraj (train)</b>			
Room 19, Cup	$-0.55 \pm 4.34$	$-9.26 \pm 0.23$	$-9.59 \pm 0.18$
Room 13, Potato	$-2.83 \pm 3.44$	$-8.01 \pm 0.58$	$-2.43 \pm 3.35$
Room 2, Lettuce	$-1.78 \pm 3.19$	$-9.30 \pm 0.37$	$-5.43 \pm 2.72$
<b>Gen-MultiTraj (test)</b>			
Room 19, Cup	$-6.39 \pm 3.20$	$-9.46 \pm 0.13$	$-3.06 \pm 3.27$
Room 13, Potato	$-2.57 \pm 2.67$	$-8.72 \pm 0.23$	$-5.61 \pm 2.93$
Room 2, Lettuce	$-3.06 \pm 2.86$	$-9.10 \pm 0.33$	$-2.99 \pm 3.40$

Table 10: Comparison of final performance (mean $\pm$ SEM) between methods in the environments for CHORES.

## C.11 FUTURE WORK

CORA is an extensive, on-going effort which we continue to develop and maintain. We are working on integrating in more baselines, such as a modular method and a continual supervised learning method adapted to the RL domain. We also hope to incorporate other useful metrics, such as for measuring data efficiency. While the set of benchmarks we present already pose considerable difficulty, we see several directions to build off CORA for more challenging evaluations. For instance, Procgen, MiniHack, and CHORES could be scaled up to even longer and more diverse task sequences. Agents could also be trained on tasks from a mix of different environments.

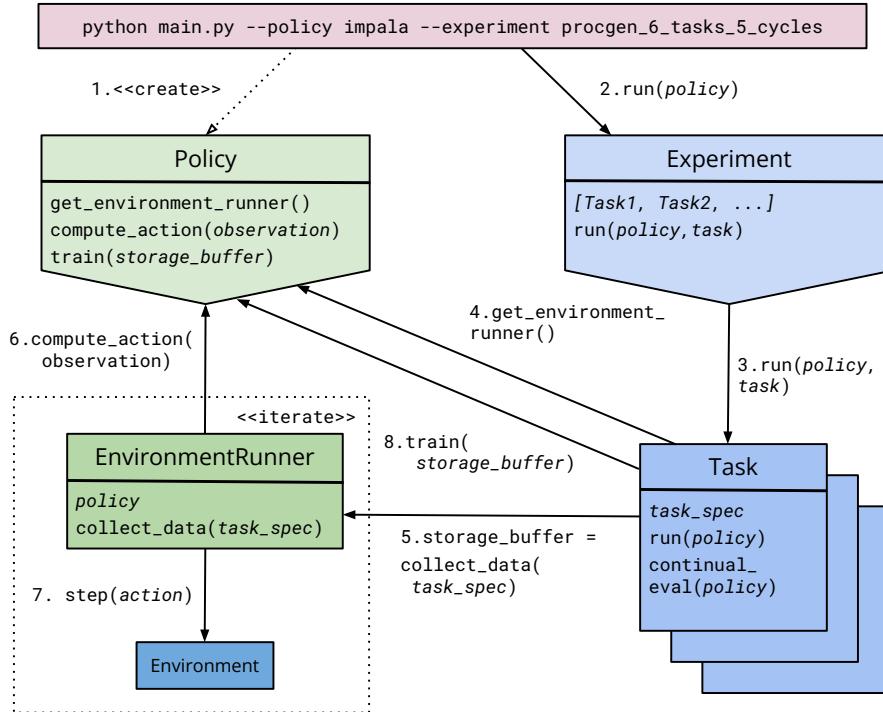


Figure 12: Sequence diagram representing the most basic flow of the `continual_rl` package. Blue represents components defined by `Experiment`, and green represents components defined by `Policy`.

## D CODE STRUCTURE

### D.1 ARCHITECTURE DIAGRAM

An overview of our code package architecture can be seen in Figure 12. The two fundamental components of the package are `Experiment` and `Policy`. `Experiment` conceptually encapsulates everything that should remain the same between runs, such as task specification, ordering, duration, and observation dimensionality. `Policy` encapsulates everything an algorithm has control over and can change as tasks are learned.

To train and evaluate agents on our benchmarks, these two things must be specified. They may be specified either via command line, or by configuration file, along with any hyperparameter changes from the defaults. We recommend referring to the README provided with the source code for more details on running experiments and implementing new policies and experiments.

### D.2 POLICIES

Any `Policy` must implement: (i) computing an action given an observation and (ii) training in response to collected experience. Any existing code that does these can be integrated into the `continual_rl` package by implementing a simple adapter wrapper. We provide an example of doing this for PPO based on the `pytorch-a2c-ppo-acktr` repository ([Kostrikov, 2018](#)). This enables easier integration of agents from outside our codebase, so the experiments and metrics provided by `continual_rl` can be leveraged.

The other thing a `Policy` must specify is how it should be run (i.e. its training loop), which we encapsulate in modules we refer to as `EnvironmentRunners`. In most simple cases, an existing `EnvironmentRunner` will suffice, such as `EnvironmentRunnerBatch` for standard, synchronous RL. However, in highly asynchronous or distributed cases, a user of `CORA` may wish to write their own. We provide more details on `EnvironmentRunners` in Section D.2.1.

The final two steps to using a `policy` in `CORA` are the specification of configuration parameters, by extending `ConfigBase`, and adding the new policy to `continual_rl/available_policies.py`, so it can be used identically to existing ones, either via config file or via command line.

Additionally, since the policies are independent modules, it is also easy to use the provided policy implementations in a separate code base. The framework is installable as a pip package, which can be imported directly.

#### D.2.1 ENVIRONMENTRUNNERS

EnvironmentRunners have one function they must implement: `collect_data()`. Given the task specification, the EnvironmentRunner must collect any number of steps worth of data from the environment and return the results of what it has collected. The function will be called repeatedly until the total number of steps for the task have been satisfied. Data collection for continual evaluation occurs between calls to `collect_data()`, so care should be taken when selecting how much data to collect at a time. If too many timesteps are collected at once, the metrics will not be able to be computed as often as desired.

EnvironmentRunners can also be viewed as a higher-level API for more advanced policies. One example of how this is useful is for IMPALA (Espeholt et al., 2018). IMPALA’s key feature is how it learns asynchronously by decoupling collecting data with actors from training policies, so the simple Policy structure of `compute_action()` and `train()` are insufficient. Instead, we define ImpalaEnvironmentRunner and implement a custom `collect_data()` method that returns new results that have accumulated every fixed number of seconds to support the actors and learners working asynchronously.

### D.3 EXPERIMENTS

Any Experiment defines a sequence of tasks. Every task contains full specifications (available in `continual_rl/task_spec.py`) for what environment should be created, how many frames it is given as a budget, and so on. Each task also provides common preprocessing features for convenience. For instance, we can define an ImageTask that scales the observation image, stacks frames, and converts the observation to a PyTorch tensor.

Experiments use this sequence of tasks to handle collecting metrics such as the Continual Evaluation metric described in Section 4.1. The Forgetting and Transfer metrics are computed in a post-processing step using the collected data from continual evaluation.

## E METRICS

Recall that in all cases, each table represents how much training on the task in each column impacts the performance of the task in each row. These metrics are also only computed across the first cycle of the task sequence.

### E.1 STANDARD ERROR OF THE MEAN

We present the standard error of the mean in our detailed metric tables. To aggregate metrics across rows and columns, we use the following procedure. First, we define a set of individual metric values  $\mathbb{M}_{i,j} = \{\mathcal{M}_{i,j,s} : s \in S\}$ , where  $S$  represents the set of seeds used,  $\mathcal{M}$  represents the metric to compute (either  $\mathcal{F}$  or  $\mathcal{Z}$ ), and  $i$  and  $j$  are task ids as defined in Section 4.1. For each non-aggregate entry in the table we compute  $SEM_{i,j} = \frac{\sigma(\mathbb{M}_{i,j}, ddof=1)}{\sqrt{|\mathbb{M}_{i,j}|}}$  as usual, where  $\sigma(\mathbb{M})$  and  $|\mathbb{M}|$  are the standard deviation and size of set  $\mathbb{M}$ .

However, for the aggregate values (row, column, and table averages), it is not the case that  $\mathbb{M}_{i,j}$  is independent from  $\mathbb{M}_{i+1,j}$  or  $\mathbb{M}_{i,j+1}$ , so we cannot simply create an aggregate set across the rows or columns. Instead, we compute  $\mathcal{M}_{j,s} = \frac{1}{|j-i|} \sum_i \mathcal{M}_{i,j,s}$ . This metric averages a given seed over all  $i$ , which yields a metric that is independent and can be aggregated as described above. This allows us to define  $SEM_j = \frac{\sigma(\mathbb{M}_j, ddof=1)}{\sqrt{N}}$ , where  $N$  is the number of tasks.  $SEM_i$  is defined symmetrically. Finally, the full-table-aggregate  $SEM$  is computed by averaging over the full set of  $i \cdot j$  entries in  $\mathcal{M}$ .

### E.2 ATARI METRICS: FORGETTING

	1-Krull	2-BeamRider	3-Hero	4-StarGunner	5-MsPacman	Avg ± SEM		1-Krull	2-BeamRider	3-Hero	4-StarGunner	5-MsPacman	Avg ± SEM
0-SpaceInvaders	3.8 ± 0.3	-0.1 ± 0.2	-0.3 ± 0.3	1.0 ± 0.4	-0.3 ± 0.3	0.8 ± 0.1		2.3 ± 1.8	-0.3 ± 0.5	0.0 ± 0.5	0.1 ± 1.0	0.4 ± 0.8	0.5 ± 0.3
1-Krull	-	5.6 ± 0.7	1.4 ± 0.5	-1.4 ± 0.5	1.0 ± 0.5	1.7 ± 0.2		-	0.6 ± 2.3	-0.5 ± 0.4	1.9 ± 1.5	-0.1 ± 0.4	0.5 ± 0.6
2-BeamRider	-	-	6.3 ± 0.2	2.1 ± 0.3	-0.0 ± 0.1	2.8 ± 0.1		-	-	-0.3 ± 0.1	1.0 ± 0.7	-0.5 ± 0.2	0.1 ± 0.1
3-Hero	-	-	-	8.5 ± 0.2	0.0 ± 0.0	4.3 ± 0.1		-	-	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
4-StarGunner	-	-	-	-	6.7 ± 0.3	6.7 ± 0.3		-	-	-	-0.3 ± 0.3	-0.3 ± 0.3	-0.3 ± 0.3
5-MsPacman	-	-	-	-	-	-		-	-	-	-	-	-
Avg ± SEM	3.8 ± 0.3	2.7 ± 0.4	2.5 ± 0.2	2.6 ± 0.1	1.5 ± 0.2	2.3 ± 0.1		2.3 ± 1.8	0.2 ± 1.1	-0.3 ± 0.1	0.8 ± 0.6	-0.1 ± 0.2	0.3 ± 0.3

(a) IMPALA

(b) EWC

	1-Krull	2-BeamRider	3-Hero	4-StarGunner	5-MsPacman	Avg ± SEM		1-Krull	2-BeamRider	3-Hero	4-StarGunner	5-MsPacman	Avg ± SEM
0-SpaceInvaders	5.9 ± 0.6	-0.7 ± 0.6	0.3 ± 0.4	0.6 ± 0.5	-0.2 ± 0.4	1.2 ± 0.1		4.6 ± 0.7	-0.7 ± 0.6	-0.6 ± 0.2	1.6 ± 0.5	-0.1 ± 0.3	1.0 ± 0.2
1-Krull	-	5.7 ± 0.7	0.5 ± 0.6	1.6 ± 1.0	-0.5 ± 0.4	1.8 ± 0.2		-	4.9 ± 0.4	0.1 ± 0.5	0.3 ± 0.9	0.1 ± 0.5	1.3 ± 0.1
2-BeamRider	-	-	2.6 ± 0.4	0.5 ± 0.2	0.6 ± 0.2	1.2 ± 0.1		-	-	2.4 ± 0.9	4.5 ± 0.9	-0.1 ± 0.1	2.3 ± 0.2
3-Hero	-	-	-	6.0 ± 1.6	1.7 ± 1.5	3.9 ± 0.1		-	-	5.2 ± 0.6	0.0 ± 0.0	2.6 ± 0.3	
4-StarGunner	-	-	-	-	0.1 ± 0.1	0.1 ± 0.1		-	-	-	4.7 ± 0.9	4.7 ± 0.9	
5-MsPacman	-	-	-	-	-	-		-	-	-	-	-	
Avg ± SEM	5.9 ± 0.6	2.5 ± 0.4	1.1 ± 0.3	2.2 ± 0.7	0.3 ± 0.3	1.6 ± 0.1		4.6 ± 0.7	2.1 ± 0.3	0.6 ± 0.3	2.9 ± 0.5	0.9 ± 0.2	1.8 ± 0.1

(c) Online EWC

(d) P&amp;C

	1-Krull	2-BeamRider	3-Hero	4-StarGunner	5-MsPacman	Avg ± SEM		1-Krull	2-BeamRider	3-Hero	4-StarGunner	5-MsPacman	Avg ± SEM
0-SpaceInvaders	3.6 ± 0.3	-0.7 ± 0.1	0.5 ± 0.2	0.1 ± 0.1	0.2 ± 0.1	0.7 ± 0.1		4.6 ± 0.7	-0.7 ± 0.6	-0.6 ± 0.2	1.6 ± 0.5	-0.1 ± 0.3	1.0 ± 0.2
1-Krull	-	1.9 ± 0.5	-0.1 ± 0.2	-0.2 ± 0.2	0.3 ± 0.1	0.5 ± 0.1		-	4.9 ± 0.4	0.1 ± 0.5	0.3 ± 0.9	0.1 ± 0.5	1.3 ± 0.1
2-BeamRider	-	-	1.6 ± 0.3	-0.8 ± 0.1	1.3 ± 0.3	0.7 ± 0.2		-	-	2.4 ± 0.9	4.5 ± 0.9	-0.1 ± 0.1	2.3 ± 0.2
3-Hero	-	-	-	0.2 ± 0.3	1.7 ± 1.0	1.0 ± 0.4		-	-	5.2 ± 0.6	0.0 ± 0.0	2.6 ± 0.3	
4-StarGunner	-	-	-	-	-	-		-	-	-	4.7 ± 0.9	4.7 ± 0.9	
5-MsPacman	-	-	-	-	-	-		-	-	-	-	-	
Avg ± SEM	3.6 ± 0.3	0.6 ± 0.3	0.7 ± 0.2	-0.2 ± 0.1	1.0 ± 0.3	0.7 ± 0.1		4.6 ± 0.7	2.1 ± 0.3	0.6 ± 0.3	2.9 ± 0.5	0.9 ± 0.2	1.8 ± 0.1

(e) CLEAR

Table 11: Atari Forgetting metrics with the standard error. We can see that forgetting across the board is quite high, though mitigated by both EWC and CLEAR.

### E.3 ATARI METRICS: TRANSFER

	0-SpaceInvaders	1-Krull	2-BeamRider	3-Hero	4-StarGunner	Avg ± SEM		0-SpaceInvaders	1-Krull	2-BeamRider	3-Hero	4-StarGunner	Avg ± SEM
0-SpaceInvaders	-	-	-	-	-	-	0-SpaceInvaders	-	-	-	-	-	-
1-Krull	0.1 ± 0.0	-	-	-	-	0.1 ± 0.0	1-Krull	1.9 ± 1.2	-	-	-	-	1.9 ± 1.2
2-BeamRider	0.2 ± 0.2	0.0 ± 0.0	-	-	-	0.1 ± 0.1	2-BeamRider	0.2 ± 2.0	-2.7 ± 1.3	-	-	-	-1.3 ± 1.4
3-Hero	0.0 ± 0.0	-0.0 ± 0.0	0.2 ± 0.2	-	-	0.1 ± 0.1	3-Hero	0.0 ± 0.0	-0.0 ± 0.0	0.0 ± 0.0	-	-	0.0 ± 0.0
4-StarGunner	0.0 ± 0.0	-0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	-	0.0 ± 0.0	4-StarGunner	3.3 ± 1.2	-1.7 ± 2.0	-0.1 ± 1.1	0.3 ± 0.6	-	0.4 ± 0.5
5-MsPacman	0.6 ± 0.2	-0.4 ± 0.3	0.7 ± 0.3	-0.8 ± 0.4	0.2 ± 0.5	0.1 ± 0.1	5-MsPacman	1.6 ± 0.5	-1.4 ± 0.4	0.8 ± 0.7	-0.2 ± 0.2	0.1 ± 0.5	0.2 ± 0.1
Avg ± SEM	0.2 ± 0.1	-0.1 ± 0.1	0.3 ± 0.1	-0.4 ± 0.2	0.2 ± 0.5	0.1 ± 0.0	Avg ± SEM	1.4 ± 0.4	-1.5 ± 0.4	0.2 ± 0.6	0.1 ± 0.3	0.1 ± 0.5	0.1 ± 0.2

(a) IMPALA

(b) EWC

	0-SpaceInvaders	1-Krull	2-BeamRider	3-Hero	4-StarGunner	Avg ± SEM		0-SpaceInvaders	1-Krull	2-BeamRider	3-Hero	4-StarGunner	Avg ± SEM
0-SpaceInvaders	-	-	-	-	-	-	0-SpaceInvaders	-	-	-	-	-	-
1-Krull	-0.2 ± 0.2	-	-	-	-	-0.2 ± 0.2	1-Krull	0.4 ± 0.3	-	-	-	-	0.4 ± 0.3
2-BeamRider	0.1 ± 0.5	-0.7 ± 0.3	-	-	-	-0.3 ± 0.2	2-BeamRider	0.3 ± 0.3	0.0 ± 0.1	-	-	-	0.2 ± 0.1
3-Hero	0.3 ± 0.2	-0.3 ± 0.2	0.0 ± 0.0	-	-	0.0 ± 0.0	3-Hero	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	-	-	0.0 ± 0.0
4-StarGunner	0.0 ± 0.0	-0.0 ± 0.0	0.0 ± 0.0	-0.0 ± 0.0	-	0.0 ± 0.0	4-StarGunner	-0.0 ± 0.0	0.0 ± 0.0	-0.0 ± 0.0	0.0 ± 0.0	-	0.0 ± 0.0
5-MsPacman	0.8 ± 0.3	-0.9 ± 0.4	0.2 ± 0.1	0.4 ± 0.3	-0.4 ± 0.3	0.0 ± 0.1	5-MsPacman	-0.7 ± 0.5	-0.4 ± 0.2	1.6 ± 0.4	-1.0 ± 0.4	-0.6 ± 0.3	-0.2 ± 0.1
Avg ± SEM	0.2 ± 0.1	-0.5 ± 0.1	0.1 ± 0.0	0.2 ± 0.2	-0.4 ± 0.3	-0.0 ± 0.0	Avg ± SEM	-0.0 ± 0.2	-0.1 ± 0.0	0.5 ± 0.1	-0.5 ± 0.2	-0.6 ± 0.3	-0.0 ± 0.1

(c) Online EWC

(d) P&amp;C

	0-SpaceInvaders	1-Krull	2-BeamRider	3-Hero	4-StarGunner	Avg ± SEM		0-SpaceInvaders	1-Krull	2-BeamRider	3-Hero	4-StarGunner	Avg ± SEM
0-SpaceInvaders	-	-	-	-	-	-	0-SpaceInvaders	-	-	-	-	-	-
1-Krull	0.3 ± 0.3	-	-	-	-	-	1-Krull	0.4 ± 0.3	-	-	-	-	0.4 ± 0.3
2-BeamRider	-0.3 ± 0.3	0.2 ± 0.2	-	-	-	-	2-BeamRider	0.3 ± 0.3	0.0 ± 0.1	-	-	-	0.2 ± 0.1
3-Hero	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.0	-	-	-	3-Hero	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	-	-	0.0 ± 0.0
4-StarGunner	0.0 ± 0.0	-0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	-	0.0 ± 0.0	4-StarGunner	-0.0 ± 0.0	0.0 ± 0.0	-0.0 ± 0.0	0.0 ± 0.0	-	0.0 ± 0.0
5-MsPacman	0.5 ± 0.2	-0.3 ± 0.2	0.3 ± 0.2	-0.1 ± 0.3	-0.1 ± 0.1	0.0 ± 0.0	5-MsPacman	-0.7 ± 0.5	-0.4 ± 0.2	1.6 ± 0.4	-1.0 ± 0.4	-0.6 ± 0.3	-0.2 ± 0.1
Avg ± SEM	0.1 ± 0.1	-0.0 ± 0.0	0.1 ± 0.1	-0.1 ± 0.1	-0.1 ± 0.1	0.0 ± 0.0	Avg ± SEM	-0.0 ± 0.2	-0.1 ± 0.0	0.5 ± 0.1	-0.5 ± 0.2	-0.6 ± 0.3	-0.0 ± 0.1

(e) CLEAR

Table 12: Atari Transfer metrics with standard error. Note that overall there is little forward transfer observed; this is expected because the tasks bear little similarity.

## E.4 PROCGEN METRICS: FORGETTING

	1-Dodgeball	2-Ninja	3-Starpilot	4-Bigfish	5-Fruitbot	Avg ± SEM		1-Dodgeball	2-Ninja	3-Starpilot	4-Bigfish	5-Fruitbot	Avg ± SEM
0-Climber	2.6 ± 0.2	-1.2 ± 0.2	1.4 ± 0.2	-0.7 ± 0.1	0.3 ± 0.1	0.5 ± 0.0	0-Climber	1.2 ± 0.1	-0.6 ± 0.1	0.7 ± 0.1	0.0 ± 0.0	-0.1 ± 0.0	0.2 ± 0.0
1-Dodgeball	-	5.4 ± 0.2	-0.1 ± 0.3	0.8 ± 0.2	0.1 ± 0.1	1.5 ± 0.1	1-Dodgeball	-	5.1 ± 0.4	-0.5 ± 0.3	0.9 ± 0.3	0.2 ± 0.2	1.4 ± 0.1
2-Ninja	-	-	6.2 ± 0.2	-0.9 ± 0.2	0.6 ± 0.1	2.0 ± 0.1	2-Ninja	-	-	4.3 ± 0.3	0.1 ± 0.1	-0.2 ± 0.1	1.4 ± 0.1
3-Starpilot	-	-	-	1.1 ± 0.1	0.1 ± 0.0	0.6 ± 0.0	3-Starpilot	-	-	-	-0.5 ± 0.2	0.3 ± 0.2	-0.1 ± 0.1
4-Bigfish	-	-	-	-	1.9 ± 0.2	1.9 ± 0.2	4-Bigfish	-	-	-	-	-0.2 ± 0.2	-0.2 ± 0.2
5-Fruitbot	-	-	-	-	-	-	5-Fruitbot	-	-	-	-	-	-
Avg ± SEM	2.6 ± 0.2	2.1 ± 0.1	2.5 ± 0.2	0.1 ± 0.1	0.6 ± 0.1	1.2 ± 0.0	Avg ± SEM	1.2 ± 0.1	2.2 ± 0.2	1.5 ± 0.1	0.1 ± 0.1	-0.0 ± 0.1	0.7 ± 0.0

(a) IMPALA							(b) EWC						
0-Climber	4.6 ± 0.2	-3.3 ± 0.3	3.4 ± 0.3	-0.6 ± 0.2	-0.8 ± 0.3	0.7 ± 0.1	0-Climber	0.5 ± 0.3	1.0 ± 0.5	-0.4 ± 0.2	1.1 ± 0.4	-0.5 ± 0.2	0.3 ± 0.1
1-Dodgeball	-	6.1 ± 0.3	-0.8 ± 0.2	1.0 ± 0.2	0.2 ± 0.1	1.6 ± 0.1	1-Dodgeball	-	-0.2 ± 0.2	3.0 ± 0.2	-0.2 ± 0.2	0.4 ± 0.2	0.7 ± 0.1
2-Ninja	-	-	4.7 ± 0.3	-0.6 ± 0.2	0.0 ± 0.2	1.4 ± 0.1	2-Ninja	-	-	-0.6 ± 0.1	2.2 ± 0.3	-0.1 ± 0.0	0.5 ± 0.1
3-Starpilot	-	-	-	0.5 ± 0.2	1.4 ± 0.1	0.9 ± 0.1	3-Starpilot	-	-	-	-0.6 ± 0.2	1.7 ± 0.2	0.5 ± 0.1
4-Bigfish	-	-	-	-	0.1 ± 0.2	0.1 ± 0.2	4-Bigfish	-	-	-	-	-0.2 ± 0.2	-0.2 ± 0.2
5-Fruitbot	-	-	-	-	-	-	5-Fruitbot	-	-	-	-	-	-
Avg ± SEM	4.6 ± 0.2	1.4 ± 0.2	2.4 ± 0.2	0.1 ± 0.1	0.2 ± 0.1	1.1 ± 0.0	Avg ± SEM	0.5 ± 0.3	0.4 ± 0.2	0.6 ± 0.1	0.6 ± 0.2	0.3 ± 0.1	0.5 ± 0.0

(c) Online EWC							(d) P&C						
0-Climber	-0.4 ± 0.2	0.2 ± 0.1	0.9 ± 0.2	-0.1 ± 0.2	0.1 ± 0.1	0.2 ± 0.0	0-Climber	-0.4 ± 0.2	0.3 ± 0.2	-0.1 ± 0.1	0.3 ± 0.2	-0.1 ± 0.1	0.1 ± 0.0
1-Dodgeball	-	-0.0 ± 0.2	0.3 ± 0.2	-0.1 ± 0.1	0.3 ± 0.2	0.1 ± 0.0	1-Dodgeball	-	-0.6 ± 0.1	2.2 ± 0.3	-0.1 ± 0.0	0.5 ± 0.1	-0.1 ± 0.0
2-Ninja	-	-	-0.1 ± 0.3	-0.5 ± 0.4	0.6 ± 0.3	0.0 ± 0.1	2-Ninja	-	-	-0.6 ± 0.2	1.7 ± 0.2	0.5 ± 0.1	-0.1 ± 0.0
3-Starpilot	-	-	-	-1.6 ± 0.1	-0.4 ± 0.1	-1.0 ± 0.0	3-Starpilot	-	-	-	-0.2 ± 0.2	-0.2 ± 0.2	-0.2 ± 0.2
4-Bigfish	-	-	-	-	0.2 ± 0.1	0.2 ± 0.1	4-Bigfish	-	-	-	-	-	-
5-Fruitbot	-	-	-	-	-	-	5-Fruitbot	-	-	-	-	-	-
Avg ± SEM	-0.4 ± 0.2	0.1 ± 0.1	0.4 ± 0.1	-0.5 ± 0.1	0.2 ± 0.1	-0.0 ± 0.0	Avg ± SEM	-0.4 ± 0.2	0.3 ± 0.2	-0.1 ± 0.1	0.3 ± 0.2	-0.1 ± 0.1	0.1 ± 0.0

(e) CLEAR													
0-Climber	-0.4 ± 0.2	0.2 ± 0.1	0.9 ± 0.2	-0.1 ± 0.2	0.1 ± 0.1	0.2 ± 0.0	0-Climber	-0.4 ± 0.2	0.3 ± 0.2	-0.1 ± 0.1	0.3 ± 0.2	-0.1 ± 0.1	0.1 ± 0.0
1-Dodgeball	-	-0.0 ± 0.2	0.3 ± 0.2	-0.1 ± 0.1	0.3 ± 0.2	0.1 ± 0.0	1-Dodgeball	-	-0.6 ± 0.1	2.2 ± 0.3	-0.1 ± 0.0	0.5 ± 0.1	-0.1 ± 0.0
2-Ninja	-	-	-0.1 ± 0.3	-0.5 ± 0.4	0.6 ± 0.3	0.0 ± 0.1	2-Ninja	-	-	-0.6 ± 0.2	1.7 ± 0.2	0.5 ± 0.1	-0.1 ± 0.0
3-Starpilot	-	-	-	-1.6 ± 0.1	-0.4 ± 0.1	-1.0 ± 0.0	3-Starpilot	-	-	-	-0.2 ± 0.2	-0.2 ± 0.2	-0.2 ± 0.2
4-Bigfish	-	-	-	-	0.2 ± 0.1	0.2 ± 0.1	4-Bigfish	-	-	-	-	-	-
5-Fruitbot	-	-	-	-	-	-	5-Fruitbot	-	-	-	-	-	-
Avg ± SEM	-0.4 ± 0.2	0.1 ± 0.1	0.4 ± 0.1	-0.5 ± 0.1	0.2 ± 0.1	-0.0 ± 0.0	Avg ± SEM	-0.4 ± 0.2	0.3 ± 0.2	-0.1 ± 0.1	0.3 ± 0.2	-0.1 ± 0.1	0.1 ± 0.0

Table 13: Progen Forgetting metrics

## E.5 PROCGEN METRICS: TRANSFER

	0-Climber	1-Dodgeball	2-Ninja	3-Starpilot	4-Bigfish	Avg ± SEM		0-Climber	1-Dodgeball	2-Ninja	3-Starpilot	4-Bigfish	Avg ± SEM
0-Climber	–	–	–	–	–	–	0-Climber	–	–	–	–	–	–
1-Dodgeball	0.0 ± 0.0	–	–	–	–	0.0 ± 0.0	1-Dodgeball	-0.4 ± 0.3	–	–	–	–	-0.4 ± 0.3
2-Ninja	3.0 ± 0.8	-4.2 ± 0.5	–	–	–	-0.6 ± 0.3	2-Ninja	2.3 ± 0.7	-3.3 ± 0.4	–	–	–	-0.5 ± 0.2
3-Starpilot	-0.0 ± 0.0	0.3 ± 0.0	-0.3 ± 0.0	–	–	0.0 ± 0.0	3-Starpilot	-1.3 ± 0.4	3.5 ± 0.2	-3.1 ± 0.2	–	–	-0.3 ± 0.2
4-Bigfish	1.1 ± 0.4	-2.0 ± 0.2	0.3 ± 0.1	-0.2 ± 0.1	–	-0.2 ± 0.1	4-Bigfish	3.8 ± 0.6	-4.3 ± 0.4	1.6 ± 0.4	-1.5 ± 0.3	–	-0.1 ± 0.1
5-Fruitbot	0.1 ± 0.1	-0.2 ± 0.0	-0.1 ± 0.0	0.3 ± 0.0	-0.3 ± 0.0	-0.0 ± 0.0	5-Fruitbot	-0.1 ± 0.6	-2.1 ± 0.3	1.6 ± 0.2	-0.4 ± 0.3	0.5 ± 0.2	-0.1 ± 0.1
Avg ± SEM	0.8 ± 0.2	-1.5 ± 0.1	-0.0 ± 0.0	0.0 ± 0.0	-0.3 ± 0.0	-0.1 ± 0.0	Avg ± SEM	0.9 ± 0.2	-1.5 ± 0.1	0.0 ± 0.1	-1.0 ± 0.3	0.5 ± 0.2	-0.2 ± 0.1

	(a) IMPALA						(b) EWC						
0-Climber	–	–	–	–	–	–	0-Climber	–	–	–	–	–	
1-Dodgeball	-0.6 ± 0.5	–	–	–	–	-0.6 ± 0.5	1-Dodgeball	-0.7 ± 0.4	–	–	–	-0.7 ± 0.4	
2-Ninja	4.5 ± 0.5	-4.3 ± 0.5	–	–	–	0.1 ± 0.1	2-Ninja	1.6 ± 0.5	-0.3 ± 0.1	–	–	0.7 ± 0.2	
3-Starpilot	-0.7 ± 0.2	2.2 ± 0.0	-2.1 ± 0.1	–	–	-0.2 ± 0.1	3-Starpilot	-0.9 ± 0.3	0.8 ± 0.2	0.2 ± 0.2	–	0.0 ± 0.2	
4-Bigfish	1.8 ± 0.7	-2.7 ± 0.3	1.1 ± 0.1	-0.8 ± 0.1	–	-0.2 ± 0.1	4-Bigfish	1.4 ± 0.4	-0.7 ± 0.3	-0.7 ± 0.4	0.3 ± 0.2	–	0.1 ± 0.1
5-Fruitbot	0.3 ± 0.2	-0.5 ± 0.1	0.3 ± 0.0	-0.1 ± 0.0	-0.1 ± 0.0	-0.0 ± 0.0	5-Fruitbot	0.1 ± 0.1	-0.1 ± 0.1	-0.2 ± 0.1	0.1 ± 0.1	0.2 ± 0.1	0.0 ± 0.0
Avg ± SEM	1.1 ± 0.2	-1.3 ± 0.1	-0.2 ± 0.0	-0.5 ± 0.1	-0.1 ± 0.0	-0.1 ± 0.1	Avg ± SEM	0.3 ± 0.2	-0.1 ± 0.1	-0.3 ± 0.1	0.2 ± 0.1	0.2 ± 0.1	0.1 ± 0.1

	(c) Online EWC						(d) P&C						
0-Climber	–	–	–	–	–	–	0-Climber	–	–	–	–	–	
1-Dodgeball	-0.2 ± 0.1	–	–	–	–	-0.2 ± 0.1	1-Dodgeball	-0.7 ± 0.4	–	–	–	-0.7 ± 0.4	
2-Ninja	3.2 ± 0.5	-3.3 ± 0.3	–	–	–	-0.0 ± 0.2	2-Ninja	1.6 ± 0.5	-0.3 ± 0.1	–	–	0.7 ± 0.2	
3-Starpilot	-0.1 ± 0.0	0.1 ± 0.0	-0.0 ± 0.0	–	–	-0.0 ± 0.0	3-Starpilot	-0.9 ± 0.3	0.8 ± 0.2	0.2 ± 0.2	–	0.0 ± 0.2	
4-Bigfish	0.6 ± 0.5	-1.2 ± 0.2	0.0 ± 0.1	-0.0 ± 0.1	–	-0.1 ± 0.1	4-Bigfish	1.4 ± 0.4	-0.7 ± 0.3	-0.7 ± 0.4	0.3 ± 0.2	–	0.1 ± 0.1
5-Fruitbot	0.0 ± 0.1	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	0.0 ± 0.0	-0.0 ± 0.0	5-Fruitbot	0.1 ± 0.1	-0.1 ± 0.1	-0.2 ± 0.1	0.1 ± 0.1	0.2 ± 0.1	0.0 ± 0.0
Avg ± SEM	0.7 ± 0.1	-1.1 ± 0.1	-0.0 ± 0.0	-0.0 ± 0.0	0.0 ± 0.0	-0.1 ± 0.0	Avg ± SEM	0.3 ± 0.2	-0.1 ± 0.1	-0.3 ± 0.1	0.2 ± 0.1	0.2 ± 0.1	0.1 ± 0.1

	(e) CLEAR					
0-Climber	–	–	–	–	–	–
1-Dodgeball	-0.2 ± 0.1	–	–	–	–	-0.2 ± 0.1
2-Ninja	3.2 ± 0.5	-3.3 ± 0.3	–	–	–	-0.0 ± 0.2
3-Starpilot	-0.1 ± 0.0	0.1 ± 0.0	-0.0 ± 0.0	–	–	-0.0 ± 0.0
4-Bigfish	0.6 ± 0.5	-1.2 ± 0.2	0.0 ± 0.1	-0.0 ± 0.1	–	-0.1 ± 0.1
5-Fruitbot	0.0 ± 0.1	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	0.0 ± 0.0	-0.0 ± 0.0
Avg ± SEM	0.7 ± 0.1	-1.1 ± 0.1	-0.0 ± 0.0	-0.0 ± 0.0	0.0 ± 0.0	-0.1 ± 0.0

Table 14: Progen Transfer metrics

## E.6 MINIHack METRICS: FORGETTING

	1-R-Dar..	2-R-Mon..	3-R-Tra..	4-R-Ult..	5-Cor..	6-Cor..	7-KeyR	8-KeyR-..	9-River-Na..	10-River-M..	11-River-L..	12-HnS..	13-HnS..	14-Corrido..	Avg ± SEM
0-R-Ran..	-0.3 ± 0.3	-0.5 ± 0.1	-0.4 ± 0.4	0.0 ± 0.3	-0.0 ± 0.2	2.3 ± 0.4	1.4 ± 0.3	-0.2 ± 0.1	-0.2 ± 0.3	-0.2 ± 0.2	1.0 ± 0.5	-1.8 ± 0.5	-0.6 ± 0.4	2.4 ± 0.5	0.2 ± 0.1
1-R-Dar..	-	-0.8 ± 0.2	1.6 ± 0.5	-1.3 ± 0.5	-0.1 ± 0.3	2.5 ± 0.4	0.2 ± 0.3	-0.5 ± 0.6	0.2 ± 0.2	0.3 ± 0.4	0.6 ± 0.2	-1.6 ± 0.6	-1.0 ± 0.4	1.9 ± 0.5	0.2 ± 0.0
2-R-Mon..	-	-	-0.4 ± 0.2	-0.1 ± 0.3	-0.4 ± 0.2	3.2 ± 0.4	1.9 ± 0.4	-0.4 ± 0.2	0.2 ± 0.3	-0.2 ± 0.4	1.4 ± 0.6	-2.2 ± 0.5	-1.8 ± 0.7	2.6 ± 0.7	0.3 ± 0.1
3-R-Tra..	-	-	-	0.5 ± 0.3	-0.3 ± 0.3	2.6 ± 0.4	1.7 ± 0.4	-0.4 ± 0.2	-0.5 ± 0.4	-0.5 ± 0.3	1.4 ± 0.5	-1.3 ± 0.3	-0.1 ± 0.4	2.2 ± 0.5	0.5 ± 0.1
4-R-Ult..	-	-	-	-	0.4 ± 0.5	4.4 ± 0.8	-0.3 ± 0.6	-1.3 ± 0.6	0.2 ± 0.4	0.6 ± 0.5	0.7 ± 0.5	-1.3 ± 0.4	-1.0 ± 0.7	1.7 ± 0.4	0.4 ± 0.1
5-Cor..	-	-	-	-	-	0.3 ± 0.4	2.3 ± 0.2	0.8 ± 0.1	0.1 ± 0.1	0.1 ± 0.1	0.2 ± 0.2	-0.5 ± 0.2	-0.6 ± 0.2	-0.0 ± 0.3	0.3 ± 0.1
6-Cor..	-	-	-	-	-	-	2.3 ± 0.2	0.9 ± 0.2	-0.0 ± 0.1	0.1 ± 0.1	0.4 ± 0.1	-0.7 ± 0.2	-0.6 ± 0.2	0.1 ± 0.4	0.3 ± 0.1
7-KeyR	-	-	-	-	-	-	-0.3 ± 0.2	-0.9 ± 0.3	0.9 ± 0.3	1.4 ± 0.2	0.6 ± 0.1	0.2 ± 0.1	0.0 ± 0.0	0.3 ± 0.0	-
8-KeyR-..	-	-	-	-	-	-	-	0.4 ± 0.3	1.0 ± 0.2	0.6 ± 0.2	0.3 ± 0.1	0.1 ± 0.0	0.0 ± 0.0	0.4 ± 0.1	-
9-River-Na..	-	-	-	-	-	-	-	-	4.7 ± 1.1	-2.4 ± 0.5	2.6 ± 0.7	2.2 ± 0.5	0.9 ± 0.3	-	-0.3 ± 0.1
10-River-M..	-	-	-	-	-	-	-	-	-	-1.5 ± 0.4	0.6 ± 0.6	1.6 ± 0.4	-0.1 ± 0.5	0.2 ± 0.2	-
11-River-L..	-	-	-	-	-	-	-	-	-	-	0.5 ± 0.6	1.6 ± 0.4	-0.1 ± 0.5	0.7 ± 0.2	-
12-HnS..	-	-	-	-	-	-	-	-	-	-	-	-4.0 ± 1.0	3.7 ± 0.7	-0.1 ± 0.4	-
13-HnS..	-	-	-	-	-	-	-	-	-	-	-	-	3.9 ± 0.7	3.9 ± 0.7	-
14-Corrido..	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Avg ± SEM	-0.3 ± 0.3	-0.7 ± 0.1	0.3 ± 0.3	-0.2 ± 0.2	-0.1 ± 0.3	2.5 ± 0.4	1.4 ± 0.2	-0.2 ± 0.2	-0.0 ± 0.2	-0.3 ± 0.2	0.3 ± 0.2	-0.4 ± 0.1	-0.3 ± 0.2	1.4 ± 0.3	0.3 ± 0.0

(a) IMPALA

	1-R-Dar..	2-R-Mon..	3-R-Tra..	4-R-Ult..	5-Cor..	6-Cor..	7-KeyR	8-KeyR-..	9-River-Na..	10-River-M..	11-River-L..	12-HnS..	13-HnS..	14-Corrido..	Avg ± SEM
0-R-Ran..	-0.4 ± 0.2	-0.6 ± 0.2	-0.4 ± 0.2	0.1 ± 0.2	-0.6 ± 0.3	-0.2 ± 0.3	0.4 ± 0.2	0.0 ± 0.2	0.9 ± 0.4	1.3 ± 0.5	0.3 ± 0.2	-0.5 ± 0.2	0.3 ± 0.2	0.5 ± 0.2	0.1 ± 0.0
1-R-Dar..	-	-2.7 ± 0.3	-0.5 ± 0.3	0.2 ± 0.2	-0.6 ± 0.4	-1.2 ± 0.4	-0.2 ± 0.4	0.0 ± 0.2	-0.2 ± 0.2	0.1 ± 0.3	-0.2 ± 0.2	1.5 ± 0.4	1.5 ± 0.6	-0.5 ± 0.4	-0.2 ± 0.1
2-R-Mon..	-	-	-0.6 ± 0.2	-0.3 ± 0.2	-0.5 ± 0.3	0.0 ± 0.3	0.3 ± 0.3	-0.2 ± 0.2	1.0 ± 0.4	1.3 ± 0.5	0.5 ± 0.2	-0.8 ± 0.2	-0.1 ± 0.3	1.1 ± 0.2	0.2 ± 0.1
3-R-Tra..	-	-	-	0.2 ± 0.2	-0.6 ± 0.2	-0.4 ± 0.4	0.0 ± 0.2	0.0 ± 0.2	0.9 ± 0.4	1.1 ± 0.5	0.5 ± 0.2	-0.2 ± 0.2	0.5 ± 0.4	0.8 ± 0.2	0.2 ± 0.1
4-R-Ult..	-	-	-	-	-1.1 ± 0.2	-0.8 ± 0.3	-0.4 ± 0.2	0.1 ± 0.2	0.2 ± 0.2	0.1 ± 0.3	-0.4 ± 0.2	1.1 ± 0.3	1.9 ± 0.4	0.2 ± 0.2	0.1 ± 0.1
5-Cor..	-	-	-	-	-	0.1 ± 0.2	0.6 ± 0.1	-0.4 ± 0.2	-0.3 ± 0.2	-0.4 ± 0.2	0.1 ± 0.1	0.2 ± 0.2	0.8 ± 0.3	0.9 ± 0.2	0.2 ± 0.0
6-Cor..	-	-	-	-	-	-	0.5 ± 0.1	-0.3 ± 0.2	-0.4 ± 0.2	-0.4 ± 0.2	0.1 ± 0.1	0.2 ± 0.2	0.9 ± 0.4	0.9 ± 0.2	0.2 ± 0.0
7-KeyR	-	-	-	-	-	-	-	0.3 ± 0.3	0.4 ± 0.1	0.4 ± 0.2	0.3 ± 0.2	-0.2 ± 0.1	-0.1 ± 0.3	-0.4 ± 0.2	0.1 ± 0.1
8-KeyR-..	-	-	-	-	-	-	-	-	0.7 ± 0.2	0.3 ± 0.1	-0.1 ± 0.0	0.0 ± 0.0	0.1 ± 0.1	-0.0 ± 0.0	0.2 ± 0.0
9-River-Na..	-	-	-	-	-	-	-	-	-	-1.5 ± 0.3	-0.8 ± 0.2	0.8 ± 0.2	1.1 ± 0.2	0.4 ± 0.2	-0.0 ± 0.1
10-River-M..	-	-	-	-	-	-	-	-	-	-	0.1 ± 0.4	0.2 ± 0.4	0.7 ± 0.1	0.8 ± 0.1	0.5 ± 0.2
11-River-L..	-	-	-	-	-	-	-	-	-	-	-	0.2 ± 0.3	0.9 ± 0.1	0.7 ± 0.1	0.6 ± 0.1
12-HnS..	-	-	-	-	-	-	-	-	-	-	-	-5.2 ± 0.4	2.6 ± 0.7	-1.3 ± 0.4	-
13-HnS..	-	-	-	-	-	-	-	-	-	-	-	-	2.5 ± 0.5	2.5 ± 0.5	-
14-Corrido..	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Avg ± SEM	-0.4 ± 0.2	-1.7 ± 0.2	-0.5 ± 0.2	0.0 ± 0.1	-0.7 ± 0.2	-0.4 ± 0.2	0.2 ± 0.1	-0.1 ± 0.1	0.3 ± 0.1	0.2 ± 0.1	0.0 ± 0.1	0.2 ± 0.1	0.3 ± 0.1	0.8 ± 0.1	0.1 ± 0.0

(b) CLEAR

### E.7 MINIHACK METRICS: TRANSFER

	0-R-Ran..	1-R-Dar..	2-R-Mon..	3-R-Tra..	4-R-Ult..	5-Cor..	6-Cor..	7-KeyR	8-KeyR-..	9-River-Na..	10-River-M..	11-River-L..	12-HideNSE..	13-HideNSE..	Avg ± SEM
0-R-Ran..	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
1-R-Dar..	4.0 ± 0.3	—	—	—	—	—	—	—	—	—	—	—	—	—	4.0 ± 0.3
2-R-Mon..	8.6 ± 1.1	0.2 ± 0.3	—	—	—	—	—	—	—	—	—	—	—	—	4.4 ± 0.5
3-R-Tra..	7.5 ± 0.9	0.9 ± 0.4	1.2 ± 0.5	—	—	—	—	—	—	—	—	—	—	—	3.2 ± 0.4
4-R-Ult..	5.8 ± 0.4	3.0 ± 0.6	1.1 ± 0.4	-2.2 ± 0.7	—	—	—	—	—	—	—	—	—	—	1.9 ± 0.2
5-Cor..	3.2 ± 0.7	-0.1 ± 0.2	0.4 ± 0.2	0.2 ± 0.4	-0.4 ± 0.4	—	—	—	—	—	—	—	—	—	0.7 ± 0.1
6-Cor..	3.2 ± 0.7	-0.1 ± 0.2	0.5 ± 0.3	0.2 ± 0.4	-0.5 ± 0.4	1.4 ± 0.3	—	—	—	—	—	—	—	—	0.8 ± 0.0
7-KeyR	1.1 ± 0.4	0.3 ± 0.2	1.3 ± 0.4	0.9 ± 0.5	-0.2 ± 0.5	2.3 ± 0.4	-0.4 ± 0.3	—	—	—	—	—	—	—	0.8 ± 0.0
8-KeyR-..	3.1 ± 0.8	-1.0 ± 0.5	0.5 ± 0.4	1.5 ± 0.5	-1.3 ± 0.4	1.2 ± 0.3	-0.1 ± 0.3	-2.1 ± 0.3	—	—	—	—	—	—	0.2 ± 0.0
9-River-Na..	6.4 ± 0.1	-0.1 ± 0.2	0.1 ± 0.1	0.4 ± 0.3	-0.5 ± 0.3	-0.4 ± 0.2	-2.0 ± 0.2	-0.7 ± 0.4	0.4 ± 0.2	—	—	—	—	—	0.4 ± 0.0
10-River-M..	7.2 ± 0.3	-0.3 ± 0.2	-0.0 ± 0.1	0.5 ± 0.3	-0.8 ± 0.2	-0.6 ± 0.2	-1.8 ± 0.2	-1.0 ± 0.4	-0.2 ± 0.2	-0.0 ± 0.4	—	—	—	—	0.3 ± 0.0
11-River-L..	7.1 ± 0.4	-0.3 ± 0.3	-0.0 ± 0.1	0.5 ± 0.3	-0.8 ± 0.2	-0.8 ± 0.3	-1.9 ± 0.2	-1.0 ± 0.4	-0.0 ± 0.2	0.1 ± 0.4	1.5 ± 0.6	—	—	—	0.4 ± 0.1
12-HideNSE..	4.7 ± 0.6	-1.5 ± 0.2	-0.8 ± 0.2	1.2 ± 0.2	-0.8 ± 0.2	-0.7 ± 0.2	-0.6 ± 0.3	-0.0 ± 0.3	0.7 ± 0.4	-0.2 ± 0.3	-0.1 ± 0.4	-0.2 ± 0.3	—	—	0.1 ± 0.1
13-HideNSE..	5.4 ± 0.3	-1.5 ± 0.2	-0.7 ± 0.2	1.2 ± 0.2	-0.9 ± 0.2	-0.8 ± 0.2	-0.5 ± 0.3	0.2 ± 0.4	0.7 ± 0.5	-0.2 ± 0.4	-0.1 ± 0.4	-0.5 ± 0.2	5.2 ± 0.9	—	0.6 ± 0.1
14-Corrido..	4.4 ± 0.9	-1.5 ± 0.3	-0.0 ± 0.3	1.4 ± 0.4	-1.5 ± 0.3	0.2 ± 0.3	0.0 ± 0.4	-0.1 ± 0.6	0.5 ± 0.2	-0.9 ± 0.7	-0.9 ± 0.2	-0.7 ± 0.4	0.1 ± 0.1	-0.5 ± 0.2	0.0 ± 0.0
Avg ± SEM	5.1 ± 0.2	-0.2 ± 0.2	0.3 ± 0.1	0.5 ± 0.3	-0.8 ± 0.2	0.2 ± 0.1	-0.9 ± 0.2	-0.7 ± 0.3	0.3 ± 0.2	-0.2 ± 0.3	0.1 ± 0.3	-0.5 ± 0.2	2.7 ± 0.5	-0.5 ± 0.2	0.6 ± 0.0

(a) IMPALA

	0-R-Ran..	1-R-Dar..	2-R-Mon..	3-R-Tra..	4-R-Ult..	5-Cor..	6-Cor..	7-KeyR	8-KeyR-..	9-River-Na..	10-River-M..	11-River-L..	12-HideNSE..	13-HideNSE..	Avg ± SEM
0-R-Ran..	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
1-R-Dar..	2.5 ± 0.5	—	—	—	—	—	—	—	—	—	—	—	—	—	2.5 ± 0.5
2-R-Mon..	7.1 ± 0.6	0.2 ± 0.2	—	—	—	—	—	—	—	—	—	—	—	—	3.6 ± 0.4
3-R-Tra..	7.0 ± 0.8	0.7 ± 0.2	0.9 ± 0.2	—	—	—	—	—	—	—	—	—	—	—	2.9 ± 0.3
4-R-Ult..	2.1 ± 0.4	2.3 ± 0.2	2.7 ± 0.3	0.2 ± 0.2	—	—	—	—	—	—	—	—	—	—	1.8 ± 0.2
5-Cor..	1.5 ± 0.5	0.2 ± 0.2	0.8 ± 0.2	0.1 ± 0.2	-0.5 ± 0.2	—	—	—	—	—	—	—	—	—	0.4 ± 0.1
6-Cor..	1.6 ± 0.5	0.1 ± 0.2	0.9 ± 0.2	0.2 ± 0.2	-0.6 ± 0.2	0.9 ± 0.1	—	—	—	—	—	—	—	—	0.5 ± 0.1
7-KeyR	1.3 ± 0.6	0.0 ± 0.2	1.0 ± 0.4	0.1 ± 0.2	-0.9 ± 0.3	1.5 ± 0.3	0.6 ± 0.3	—	—	—	—	—	—	—	0.5 ± 0.1
8-KeyR-..	2.2 ± 0.8	-0.9 ± 0.4	-1.0 ± 0.4	-0.2 ± 0.1	-0.0 ± 0.0	-0.1 ± 0.1	-0.4 ± 0.2	-0.1 ± 0.1	0.4 ± 0.2	—	—	—	—	—	0.1 ± 0.0
9-River-Na..	2.5 ± 1.5	0.5 ± 0.2	0.2 ± 0.2	-0.1 ± 0.1	-0.0 ± 0.2	-0.1 ± 0.1	-0.4 ± 0.2	-0.1 ± 0.1	0.4 ± 0.2	—	—	—	—	—	0.3 ± 0.2
10-River-M..	3.0 ± 1.5	0.2 ± 0.1	-0.1 ± 0.2	-0.3 ± 0.2	-0.1 ± 0.2	-0.5 ± 0.2	-0.7 ± 0.2	0.1 ± 0.3	0.5 ± 0.3	-0.4 ± 0.3	—	—	—	—	0.2 ± 0.2
11-River-L..	2.8 ± 1.3	0.3 ± 0.1	0.0 ± 0.2	-0.4 ± 0.2	-0.3 ± 0.2	-0.3 ± 0.1	-0.5 ± 0.2	-0.0 ± 0.2	0.4 ± 0.3	-0.3 ± 0.3	-0.5 ± 0.4	—	—	—	0.1 ± 0.1
12-HideNSE..	2.6 ± 0.6	-0.7 ± 0.2	-0.8 ± 0.2	-0.2 ± 0.2	0.0 ± 0.1	-0.0 ± 0.1	-0.4 ± 0.2	-0.0 ± 0.1	0.6 ± 0.2	0.4 ± 0.2	0.2 ± 0.1	-0.0 ± 0.2	—	—	0.1 ± 0.1
13-HideNSE..	2.3 ± 0.7	-0.6 ± 0.2	-0.5 ± 0.2	-0.1 ± 0.2	-0.2 ± 0.2	-0.2 ± 0.1	-0.2 ± 0.2	0.0 ± 0.2	0.5 ± 0.2	0.3 ± 0.2	-0.0 ± 0.2	-0.0 ± 0.2	3.9 ± 0.5	—	0.4 ± 0.1
14-Corrido..	0.8 ± 0.5	-0.4 ± 0.2	-0.5 ± 0.2	-0.1 ± 0.0	0.0 ± 0.0	0.1 ± 0.0	0.8 ± 0.1	1.3 ± 0.3	-0.1 ± 0.1	-0.9 ± 0.3	-0.4 ± 0.2	-0.0 ± 0.2	-0.3 ± 0.2	-0.5 ± 0.2	-0.0 ± 0.0
Avg ± SEM	2.8 ± 0.5	0.1 ± 0.1	0.3 ± 0.1	-0.1 ± 0.1	-0.3 ± 0.1	0.2 ± 0.1	-0.1 ± 0.1	0.3 ± 0.1	0.4 ± 0.1	-0.2 ± 0.1	-0.2 ± 0.1	-0.0 ± 0.1	1.8 ± 0.3	-0.5 ± 0.2	0.5 ± 0.1

(b) CLEAR

Table 15: MiniHack Transfer metrics.

### E.8 CHORES METRICS: FORGETTING

	R402	R419	R423	Avg ± SEM
R402	–	0.0 ± 0.0	1.4 ± 1.4	0.7 ± 0.7
R419	–	–	-3.3 ± 3.3	-3.3 ± 3.3
R423	–	–	–	–
Avg ± SEM	–	0.0 ± 0.0	-1.0 ± 1.0	-0.6 ± 0.6

(a) EWC

	R402	R419	R423	Avg ± SEM
R402	–	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
R419	–	–	0.0 ± 0.0	0.0 ± 0.0
R423	–	–	–	–
Avg ± SEM	–	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

(b) P&amp;C

	R402	R419	R423	Avg ± SEM
R402	–	4.1 ± 2.4	-6.1 ± 3.1	-1.0 ± 1.0
R419	–	–	-3.1 ± 3.1	-3.1 ± 3.1
R423	–	–	–	–
Avg ± SEM	–	4.1 ± 2.4	-4.6 ± 2.8	-1.7 ± 1.7

(c) CLEAR

Table 16: CHORES: Mem-VaryRoom Forgetting metrics

	Hang TP	Counter	Cabinet	Avg ± SEM
Hang TP	–	-0.0 ± 0.0	0.0 ± 0.0	-0.0 ± 0.0
Counter	–	–	6.3 ± 4.3	6.3 ± 4.3
Cabinet	–	–	–	–
Avg ± SEM	–	-0.0 ± 0.0	3.1 ± 2.2	2.1 ± 1.4

(a) EWC

	Hang TP	Counter	Cabinet	Avg ± SEM
Hang TP	–	0.1 ± 0.1	-3.3 ± 3.2	-1.6 ± 1.6
Counter	–	–	-3.5 ± 3.3	-3.5 ± 3.3
Cabinet	–	–	–	–
Avg ± SEM	–	0.1 ± 0.1	-3.4 ± 3.2	-2.2 ± 2.2

(b) P&amp;C

	Hang TP	Counter	Cabinet	Avg ± SEM
Hang TP	–	4.8 ± 0.4	0.3 ± 0.2	2.5 ± 0.3
Counter	–	–	-0.5 ± 0.5	-0.5 ± 0.5
Cabinet	–	–	–	–
Avg ± SEM	–	4.8 ± 0.4	-0.1 ± 0.2	1.5 ± 0.2

(c) CLEAR

Table 17: CHORES: Mem-VaryTasks Forgetting metrics

	Fork	Knife	Spoon	Avg ± SEM
Fork	–	3.4 ± 3.0	-3.2 ± 3.2	0.1 ± 0.1
Knife	–	–	-3.2 ± 3.2	-3.2 ± 3.2
Spoon	–	–	–	–
Avg ± SEM	–	3.4 ± 3.0	-3.2 ± 3.2	-1.0 ± 1.1

(a) EWC

	Fork	Knife	Spoon	Avg ± SEM
Fork	–	-0.0 ± 0.0	3.1 ± 3.1	1.5 ± 1.5
Knife	–	–	3.0 ± 3.0	3.0 ± 3.0
Spoon	–	–	–	–
Avg ± SEM	–	-0.0 ± 0.0	3.0 ± 3.0	2.0 ± 2.0

(b) P&amp;C

	Fork	Knife	Spoon	Avg ± SEM
Fork	–	6.7 ± 3.3	0.4 ± 6.0	3.6 ± 1.8
Knife	–	–	2.9 ± 2.9	2.9 ± 2.9
Spoon	–	–	–	–
Avg ± SEM	–	6.7 ± 3.3	1.7 ± 1.9	3.4 ± 0.2

(c) CLEAR

Table 18: CHORES: Mem-VaryObjects Forgetting metrics

	R19, Cup	R13, Potat..	R02, Lettu..	Avg ± SEM
R19, Cup	–	1.9 ± 3.2	-0.2 ± 0.1	0.9 ± 1.5
R13, Potat..	–	–	0.5 ± 0.4	0.5 ± 0.4
R02, Lettu..	–	–	–	–
Avg ± SEM	–	1.9 ± 3.2	0.2 ± 0.1	0.7 ± 1.2

(a) EWC

	R19, Cup	R13, Potat..	R02, Lettu..	Avg ± SEM
R19, Cup	–	0.0 ± 0.1	-0.2 ± 0.1	-0.1 ± 0.1
R13, Potat..	–	–	-0.1 ± 0.1	-0.1 ± 0.1
R02, Lettu..	–	–	–	–
Avg ± SEM	–	0.0 ± 0.1	-0.1 ± 0.1	-0.1 ± 0.1

(b) P&amp;C

	R19, Cup	R13, Potat..	R02, Lettu..	Avg ± SEM
R19, Cup	–	1.3 ± 1.3	-1.3 ± 1.2	-0.0 ± 1.0
R13, Potat..	–	–	-0.9 ± 4.6	-0.9 ± 4.6
R02, Lettu..	–	–	–	–
Avg ± SEM	–	1.3 ± 1.3	-1.1 ± 2.9	-0.3 ± 2.1

(c) CLEAR

Table 19: CHORES: Gen-MultiTraj Forgetting metrics

## E.9 CHORES METRICS: TRANSFER

	R402	R419	R423	Avg ± SEM
R402	–	–	–	–
R419	0.0 ± 0.0	–	0.0 ± 0.0	–
R423	-0.1 ± 0.1	0.0 ± 0.0	–	-0.1 ± 0.0
Avg ± SEM	-0.1 ± 0.0	0.0 ± 0.0	–	-0.0 ± 0.0

(a) EWC

	R402	R419	R423	Avg ± SEM
R402	–	–	–	–
R419	4.8 ± 2.9	–	–	4.8 ± 2.9
R423	4.8 ± 2.9	0.0 ± 0.0	–	2.4 ± 1.4
Avg ± SEM	4.8 ± 2.9	0.0 ± 0.0	–	3.2 ± 1.9

(b) P&amp;C

	R402	R419	R423	Avg ± SEM
R402	–	–	–	–
R419	-0.1 ± 0.1	–	–	-0.1 ± 0.1
R423	-2.0 ± 2.0	-1.3 ± 1.3	–	-1.7 ± 1.7
Avg ± SEM	-1.0 ± 1.0	-1.3 ± 1.3	–	-1.1 ± 1.1

(c) CLEAR

Table 20: CHORES: Mem-VaryRoom Transfer metrics

	Hang TP	Counter	Cabinet	Avg ± SEM
Hang TP	–	–	–	–
Counter	-6.7 ± 3.3	–	–	-6.7 ± 3.3
Cabinet	-6.3 ± 3.2	1.1 ± 1.3	–	-2.6 ± 2.2
Avg ± SEM	-6.5 ± 3.3	1.1 ± 1.3	–	-4.0 ± 2.6

(a) EWC

	Hang TP	Counter	Cabinet	Avg ± SEM
Hang TP	–	–	–	–
Counter	0.2 ± 0.1	–	–	0.2 ± 0.1
Cabinet	0.2 ± 0.3	0.1 ± 0.1	–	0.1 ± 0.1
Avg ± SEM	0.2 ± 0.2	0.1 ± 0.1	–	0.2 ± 0.1

(b) P&amp;C

	Hang TP	Counter	Cabinet	Avg ± SEM
Hang TP	–	–	–	–
Counter	-7.5 ± 0.1	–	–	-7.5 ± 0.1
Cabinet	-2.2 ± 0.0	-0.0 ± 0.1	–	-1.1 ± 0.1
Avg ± SEM	-4.8 ± 0.0	-0.0 ± 0.1	–	-3.2 ± 0.0

(c) CLEAR

Table 21: CHORES: Mem-VaryTasks Transfer metrics

	Fork	Knife	Spoon	Avg ± SEM
Fork	–	–	–	–
Knife	6.4 ± 3.1	–	–	6.4 ± 3.1
Spoon	6.9 ± 3.2	-5.3 ± 2.7	–	0.8 ± 2.8
Avg ± SEM	6.6 ± 3.1	-5.3 ± 2.7	–	2.6 ± 2.9

(a) EWC

	Fork	Knife	Spoon	Avg ± SEM
Fork	–	–	–	–
Knife	9.3 ± 0.7	–	–	9.3 ± 0.7
Spoon	8.2 ± 1.9	-1.3 ± 1.3	–	3.4 ± 1.6
Avg ± SEM	8.7 ± 1.3	-1.3 ± 1.3	–	5.4 ± 1.3

(b) P&amp;C

	Fork	Knife	Spoon	Avg ± SEM
Fork	–	–	–	–
Knife	-9.4 ± 0.0	–	–	-9.4 ± 0.0
Spoon	-6.4 ± 0.5	1.8 ± 2.6	–	-2.3 ± 1.2
Avg ± SEM	-7.9 ± 0.2	1.8 ± 2.6	–	-4.6 ± 0.8

(c) CLEAR

Table 22: CHORES: Mem-VaryObjects Transfer metrics

	R19, Cup	R13, Potat..	R02, Lettu..	Avg ± SEM
R19, Cup	–	–	–	–
R13, Potat..	-3.5 ± 1.7	–	–	-3.5 ± 1.7
R02, Lettu..	-6.3 ± 2.7	-2.3 ± 2.7	–	-4.3 ± 0.4
Avg ± SEM	-4.9 ± 2.0	-2.3 ± 2.7	–	-4.0 ± 0.5

(a) EWC

	R19, Cup	R13, Potat..	R02, Lettu..	Avg ± SEM
R19, Cup	–	–	–	–
R13, Potat..	0.9 ± 0.0	–	–	0.9 ± 0.0
R02, Lettu..	0.0 ± 0.2	0.3 ± 0.2	–	0.2 ± 0.2
Avg ± SEM	0.4 ± 0.1	0.3 ± 0.2	–	0.4 ± 0.1

(b) P&amp;C

	R19, Cup	R13, Potat..	R02, Lettu..	Avg ± SEM
R19, Cup	–	–	–	–
R13, Potat..	-4.1 ± 2.1	–	–	-4.1 ± 2.1
R02, Lettu..	-9.8 ± 0.1	-0.1 ± 0.2	–	-5.0 ± 0.0
Avg ± SEM	-7.0 ± 1.0	-0.1 ± 0.2	–	-4.7 ± 0.7

(c) CLEAR

Table 23: CHORES: Gen-MultiTraj Transfer metrics