An Empirical Study of Deep Reinforcement Learning in Continuing Tasks

Anonymous authors
Paper under double-blind review

Abstract

In reinforcement learning (RL), continuing tasks refer to tasks where the agent-environment interaction is ongoing and can not be broken down into episodes. These tasks are suitable when environment resets are unavailable, agent-controlled, or predefined but where all rewards—including those beyond resets—are critical. These scenarios frequently occur in real-world applications and can not be modeled by episodic tasks. While modern deep RL algorithms have been extensively studied and well understood in episodic tasks, their behavior in continuing tasks remains underexplored. To address this gap, we provide an empirical study of several well-known deep RL algorithms using a suite of continuing task testbeds based on Mujoco and Atari environments, highlighting several key insights concerning continuing tasks. Using these testbeds, we also investigate the effectiveness of a method for improving temporal-difference-based reinforcement learning (RL) algorithms in continuing tasks by centering rewards, as introduced by Naik et al. (2024). While their work primarily focused on this method in conjunction with Q-learning, our results extend their findings by demonstrating that this method is effective across a broader range of algorithms, scales to larger tasks, and outperforms two other reward-centering approaches.

1 Introduction

Reinforcement learning (RL) tasks can generally be divided into two categories: episodic tasks and continuing tasks. In episodic tasks, the interaction between the agent and environment naturally breaks down into distinct episodes, with the environment resetting to an initial state at the end of each episode. The goal of these tasks is to maximize the expected cumulative reward within each episode. Episodic tasks are suitable when the environment can be reset, the reset conditions are predefined, and rewards beyond the reset point do not matter—such as in video games.

In contrast, continuing tasks involve ongoing agent-environment interactions where all rewards matter. Continuing tasks are well-suited for situations where the environment cannot be reset. In many real-world problems, such as inventory management, content recommendation, and portfolio management, the environment's dynamics are beyond the control of the solution designer, making environment resets impossible. Continuing tasks can also be useful when resets are possible. First, when designing reset conditions is challenging, it can be beneficial for the agent to determine when to reset. For instance, a house-cleaning robot might decide to reset its environment by requesting to be placed back on the charging dock if trapped by cables. The second scenario involves predefined reset conditions, just as in episodic tasks, but where post-reset rewards still matter. For example, when training a robot to walk, allowing the robot to learn when to fall and reset can lead to better overall performance, as it could pursue higher rewards after resetting rather than merely avoiding falling at all costs. In both scenarios, continuing tasks provide an opportunity to balance the frequency of resets and the rewards earned by choosing the cost of reset, which is a flexibility not present in episodic tasks.

Continuing tasks can also be useful in cases where the ultimate goal is to solve an episodic task. This is best exemplified by the works on the autonomous RL setting, where the goal is to address an episodic task, and the agent learns a policy to reset the environment. In this setting, the agent is trained on a special continuing task, where the main task, which is the episodic task of interest, and an auxiliary task, such as moving to the initial state, are presented in an interleaved sequence. The learned main task's policy is deployed after training. This setting can be most useful when resets are expensive, and it is possible to reach the initial state from all other states, such as in many robotic tasks. Representative works in this direction include Eysenbach et al. (2017); Sharma et al. (2021); Zhu et al. (2020) and Sharma et al. (2022).

Despite the broad applications of continuing tasks, empirical studies on deep RL algorithms in these tasks remain limited, and their unique challenges remain under-explored. Most existing empirical studies focus on demonstrating better performance of new algorithms. For instance, Zhang and Ross (2021), Ma et al. (2021), Saxena et al. (2023), and Hisaki and Ono (2024) introduced average-reward variations of popular deep RL algorithms and empirically evaluated them alongside their discounted return counterparts on continuing tasks based on the Mujoco environment (Todorov et al., 2012), highlighting improvements in performance. In addition to the Mujoco testbeds used in the above works, Platanios et al. (2020) and Zhao et al. (2022) provided new testbeds for continuing tasks. However, Platanios et al.'s (2020) testbed also presents significant partial observability, making it not suitable for isolating the challenges of continuing tasks. The testbeds presented by Zhao et al. (2022) have small discrete state and action spaces, making them primarily suitable for studying tabular algorithms. To our knowledge, only two empirical studies have explored the unique challenges that continuing tasks present to deep RL algorithms. In particular, Sharma et al. (2022) found that several RL algorithms designed for the autonomous RL setting perform significantly worse when resets are unavailable. This indicates that resets limit the range of visited states, focusing the agents around initial and goal states. Naik et al. (2024) demonstrated that in two small-scale continuing tasks (namely, Pendulum and Catch), the DQN algorithm performs poorly when using a large discount factor or when rewards share a common offset. While a large discount factor also poses challenges in episodic tasks, its effects can be masked by the finite length of episodes. Shifting rewards by a common offset can only be applied to continuing tasks, as in episodic tasks, it changes the underlying problem.

Our first contribution is an empirical study of several well-known deep RL algorithms on a suite of continuing task testbeds. The objectives of this study include understanding the challenges present in continuing tasks with different reset scenarios and the extent to which the existing deep RL algorithms address these challenges. The tested algorithms include DDPG (Lillicrap, 2015), TD3 (Fujimoto et al., 2018), SAC (Haarnoja et al., 2018), PPO (Schulman et al., 2017), and DQN (Mnih et al., 2015). The testbeds are obtained by applying simple modifications to existing episodic testbeds from Gymnasium (Towers et al., 2024) based on Mujoco and Atari environments (Bellemare et al., 2013), such as removing time-based resets and treating resets as standard transitions in the environment with some extra cost. We considered the following reset scenarios: no resets, predefined resets, and agent-controlled resets. The proposed testbeds include 15 continuous action tasks covering all these reset scenarios and six discrete action tasks with predefined resets. We did not create Atari-based testbeds without resets or with agent-controlled resets because it is not trivial to remove the predefined resets there. While some of our Mujoco testbeds are identical to those used in prior works studying average-reward algorithms (e.g., Zhang and Ross 2021), the majority differ from theirs. The code used in this study is based on the Pearl library (Zhu et al., 2023) and will be available upon the publication of this paper.

The empirical study reveals several key insights. First, the tested algorithms perform significantly worse in tasks without resets compared to those with predefined resets. We found that predefined resets help in at least two ways. One is that they limit the effective state space the agent needs to deal with. This point echoes Sharma et al.'s (2022) finding in the autonomous RL setting. The other way is that they move the agent back to an initial state when the agent fails to escape from suboptimal states due to the weak exploration ability. Second, tested algorithms in continuing testbeds with predefined resets learn policies outperforming the same algorithms in the episodic testbed variants when both policies are evaluated in the continuing testbeds. We found that better performance is achieved by choosing actions that yield higher rewards at the cost of more frequent resets. Further, increasing the reset cost reduces the number of resets and, interestingly, can even improve overall rewards, indicating that reset costs are not only problem parameters but also solution parameters. Third, when agents are given control over resets, in some cases, it can barely surpass or even be worse than random policies in tasks with predefined resets, which suggests that these tasks are quite challenging for the tested algorithms. Lastly, all algorithms perform poorly in continuing tasks with large discount factors or shared reward offsets, which is in line with Naik et al.'s (2024) findings about deep Q-learning in small-scale tasks. These findings highlight the need for careful selection of discount factors and the avoidance of reward offsets when applying these deep RL algorithms to continuing tasks.

Our second contribution is empirically showing the effectiveness of temporal-difference (TD)–based reward centering on a wide range of deep RL algorithms. Originally proposed by Naik et al. (2024), reward centering is an idea to address challenges posed by a large discount factor and a large common reward offset by subtracting an estimate of the average-reward rate from all rewards. TD-based reward centering is one approach to estimating the reward rate and is particularly beneficial for off-policy algorithms; the reward rate can be estimated using a moving average of past rewards in the on-policy setting but not in the off-policy setting. Naik et al. (2024) demonstrated its effectiveness primarily in the tabular and linear function approximation settings, with deep RL results limited to DQN on two small-scale tasks (Pendulum and Catch) and PPO, which is an on-policy algorithm, on six Mujoco tasks. We show that TD-based reward centering improves all tested algorithms on a larger scale and more diverse testbeds. Additionally, we compare TD-based reward centering with the moving average approach, despite its theoretical issues in the off-policy setting, and an approach using a set of selected reference states (Devraj and Meyn, 2021).

Empirical results demonstrate that TD-based reward centering significantly improves performance across a wide range of continuing tasks and maintains performance in others. Furthermore, algorithms incorporating TD-based reward centering are not sensitive to reward offsets. The findings related to large discount factors present a more nuanced picture compared to Naik et al.'s (2024) results on smaller tasks. While their experiments show that, with reward centering, the discount factor primarily affects the speed of learning without degrading long-term performance even as the discount factor approaches one, our results on larger scale tasks show that long-term performance still declines, albeit much less sharply than when reward centering is not employed. This suggests that even with TD-based reward centering, tuning the discount factor remains valuable, particularly in more complex tasks. Finally, while the moving-average approach is less effective than TD-based reward centering, surprisingly, it is helpful for the tested off-policy algorithms despite its theoretical limitations. The reference-state-based approach improves the tested algorithms in some tasks but hurts in others.

2 Evaluating Deep RL Algorithms on Continuing Tasks

This section evaluates several of the most well-known RL algorithms in a suite of continuing testbeds.

2.1 Testbeds without Resets

This section evaluates four continuous control algorithms (DDPG, TD3, SAC, PPO) in five continuing testbeds without resets and shows how the absence of resets poses a significant challenge to the tested algorithms.

The testbeds are based on five Mujoco environments: Swimmer, HumanoidStandup, Reacher, Pusher, and Ant. The goal of the Swimmer and Ant testbeds is to move a controlled robot forward as fast as possible. For Reacher and Pusher, the goal is to control a robot to either reach a target position or push an object to a target position. In HumanoidStandup, the goal is to make a lying Humanoid robot stand up. The episodic versions of these testbeds have been standard in RL (Towers et al., 2024). The continuing testbeds are the same as the episodic ones except for the following differences. First, the continuing testbeds do not involve time-based or state-based resets. For Reacher, we resample the target position every 50 steps while leaving the robot's arm untouched, so that the robot needs to learn to reach a new position every 50 steps. Similarly, for Pusher, everything remains the same except that the object's position is randomly sampled every 100 step. As for Ant, we increase the range of the angles at which its legs can move, so that the ant robot can recover when it flips over.

Note that we created these continuing testbeds based on environments where, except for a set of transient states, it is possible to transition from any state to any other state. This is known as the weakly communicating property in MDPs (Puterman, 2014). Without this property, no algorithm can guarantee the quality of the learned policy because the agent might enter suboptimal states, from which there is no way to escape. An example environment without this property is Mujoco's Hopper, where if the agent falls, it is unable to stand back up.

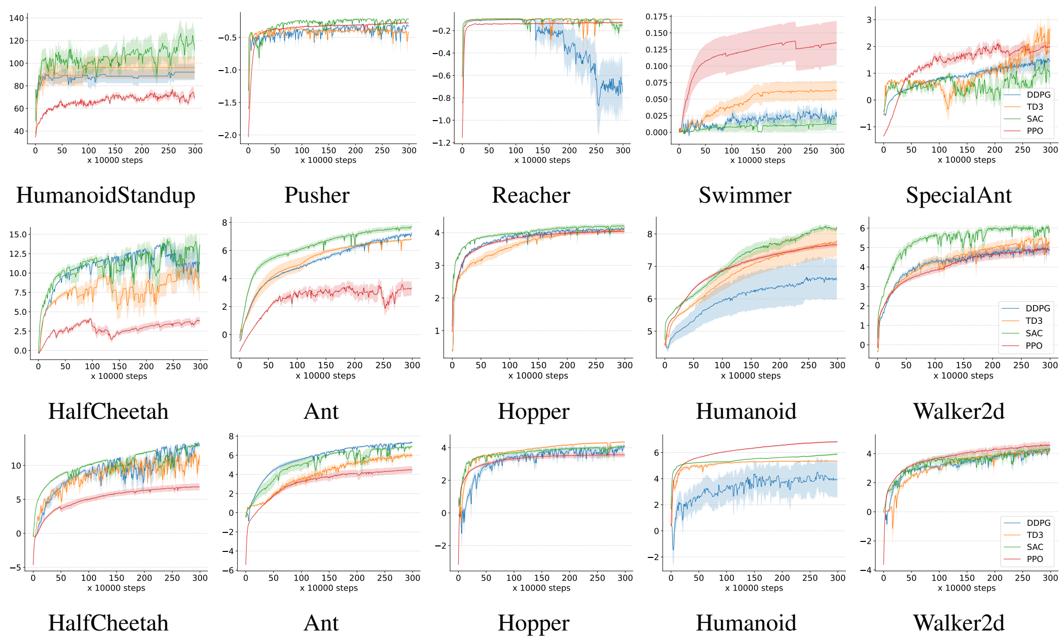

Figure 1: Learning curves in continuing testbeds without resets (upper row), with predefined resets (middle row), and with agent-controlled resets (lower row) based on the Mujoco environment. Each point in a curve shows the reward rate averaged over the past $10,000$ steps. The shading area shows one standard error.

For each task, we ran all tested algorithms for ten independent runs, with each run lasting 3 million steps. The tested parameter settings are provided in Section A.2. We report learning curves corresponding to the parameter setting that results in the highest average-reward rate across the last $10,000$ steps in the upper five plots in Figure 1. We also manually checked the learned policies by rendering videos to see if they performed reasonably well in the tested problems.

Task	DDPG	TD3	SAC	PPO
Swimmer	343.45	469.54	2428.54	29.19
HumanoidStandup	63.76	30.66	39.04	0.44
Reacher	394.67	0.02	10.48	3.42
Pusher	-4.10	1.65	-3.30	0.67
SpecialAnt	35.30	88.08	120.98	23.82

Table 1: The percentage of the final reward rate improvement when resets are applied with a small probability. The gray color indicates that the performance difference is not statistically significant. This table shows that in some tasks, the lack of resets poses a significant challenge to the tested algorithms.

For Reacher, we found that TD3 and SAC both learned descent policies in most of the runs, DDPG failed catastrophically after converging to a descent policy in half of the test runs, and PPO's learned policies did not reach the target positions across most of the runs. For Pusher, all algorithms learned policies that perform reasonably well in most of runs. For Swimmer, Humanoid-Standup, and SpecialAnt, none of the algorithms were able to learn a policy that performed reasonably well in most of the runs.

To understand if the poor performance of the tested algorithms' performance is mainly due to the unavailability of resets, we created three variants of these testbeds where resets occur with probabilities of $0.01, 0.001$, and 0.0001 per time step, respectively. Upon resetting, regardless of the current state and the chosen action, the resulting next state would be sampled from the task's initial state distribution. The reward setting and the rest of the task dynamics remain unchanged. For each resetting variant, we ran each algorithm for ten runs, each of which consists of 3 million steps. We report the percentage of improvement, defined as $\frac{\bar{r}^{\text{no resets}} - \bar{r}^{\text{random}}}{\bar{r}^{\text{random resets}} - \bar{r}^{\text{random}}} - 1$, where $r^{\text{no resets}}$ is the reward rate of the final policy learned in the task without reset, $r^{\text{random resets}}$ is the best final reward rate across all three variants with resets, and $\bar{r}^{\text{random}}$ is the reward rate of a uniformly random policy in the testbed without resets. All reward rates are averaged over ten runs. We use gray shading to indicate that the

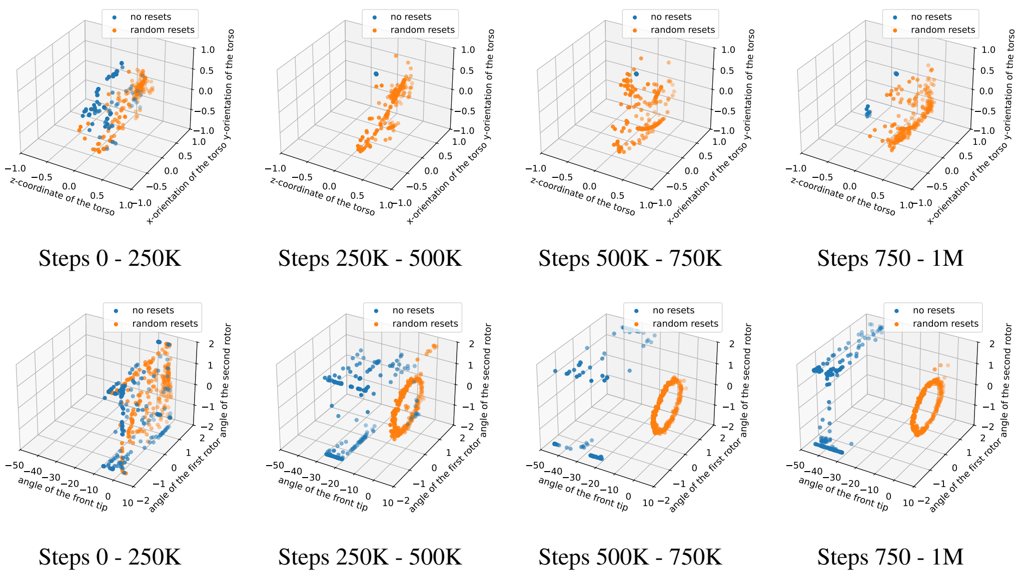

Figure 2: Evolution of DDPG's visited states in two HumanoidStandup testbeds (upper row) and TD3's visited states in two Swimmer testbeds (lower row). In both cases, one testbed does not involve resets, while the other one resets with a probability of 0.001 per time step. We visualize three key elements of the visited states in the first 1M steps of one run. For HumanoidStandup, all blue dots concentrate on a small suboptimal region, indicating that the agent fails to perform a sufficient amount of exploration without resets. For the Swimmer, the orange circle indicates the swimmer undulates like a snake to move forward, suggesting that the agent finds a decent policy. Without resetting, the agent explores a larger region of the state space but fails to learn a good policy.

reward rate difference with and without resets is not statistically significant, as determined by Welch's t-test with a p-value less than 0.05. The results (Table 1) show that, overall, the learned policies in the testbeds with random resets are significantly better than those learned in the testbeds without resets.

Visualizing the evolution of some key state elements reveals two reasons why algorithms performed much better in the reset variants of the testbeds. To illustrate these two reasons, we show in a representative run, for every 1000 steps, the evolution of the height and orientation of the Humanoid robot's torso with DDPG and the evolution of the angular component of the Swimmer robot with TD3. In both testbeds with random resets, the reset probability is 0.001. The evolution plots are shown in Figure 2. For HumanoidStandup, the agent's selected state elements concentrate on a point for a long period, suggesting that the agent is trapped in some small region in the state space. Note that the MDP is weakly communicating, therefore it is possible to move from every state to every other state. In addition, note that the z-coordinate is the main factor contributing to the task's reward. Hence, a low z-value, in general, corresponds to a low reward. Therefore, the evolution plots show that the agent did not perform sufficient exploration to escape from suboptimal states. With random resets, the exploration challenge is significantly simplified because external resets move the agent out of these suboptimal states.

Swimmer's evolution plots show that, as training progresses, the agent eventually discovers a decent policy in the reset variant of the testbed (shown by orange dots). In the original testbed, the algorithm explores a wider range of the state space but fails to converge to an effective policy (shown by blue dots). A closer look at the blue dots reveals that the front tip's angle gradually shifts from 0 to -50 rads within the first 1M steps. Notably, there is no inherent limit on how large or small this angle can be, leading the agent to continuously observe novel front tip angles that extrapolate beyond the previously encountered ones and explore ever-larger front tip angles, searching for potentially higher rewards. The testbed variant with resets avoids this challenge by constraining exploration to the vicinity of the initial state, effectively reducing the region the agent could possibly visit in the vast state space.

To verify if the limited size of the state space is indeed the main reason that explains the performance gap in Swimmer, we tested the four algorithms on a variant of the Swimmer testbed with constrained state space. This variant is only different from the original Swimmer in that the angular elements observed by the agent are converted to be within $[-\pi, \pi)$ (i.e., angle x in the original testbed is converted to $x \mod 2\pi - \pi$). Note that this conversion does not change the environment dynamics, and the new state space is equivalent to the original one. We observed that DDPG, TD3, and SAC in this new testbed achieved statistically significantly higher performance compared to the original Swimmer, with the percentage of improvement being 1233.26$\%$, 333.22$\%$ and 2287.43 $\%$. For PPO, the performance improvement is not statistically significant. The results show that constraining the state space by resets is indeed a major factor in achieving a higher performance in swimmers with resets and limiting the size of the state space can achieve similar performance gains as resets.

2.2 Testbeds with Predefined Resets

This section evaluates both continuous and discrete control algorithms on continuing task testbeds with predefined resets. In addition, it shows how the learned policies differ from policies learned in episodic variants of the testbeds.

The test suite includes both continuous and discrete control testbeds. The continuous control testbeds are built upon five Mujoco environments: HalfCheetah, Ant, Hopper, Humanoid, and Walker2d. In these testbeds, the objective is to control a simulated robot to move forward as quickly as possible. The corresponding existing episodic testbeds involve time-based truncation of the agent's experience followed by an environment reset. In the continuing testbeds, we remove this time-based truncation and reset. However, we retain state-based resets, such as when the robot is about to fall (in Hopper, Humanoid, and Walker2d) or when it flips its body (in Ant). In addition, we add a reset condition for HalfCheetah when it flips, which is not available in the existing episodic testbeds. Each reset incurs a penalty of -10 to the reward, punishing the agent for falling or flipping.

The discrete control testbeds are adapted from six Atari environments: Breakout, Pong, Space Invaders, BeamRider, Seaquest, and Ms. PacMan. Like the Mujoco environments, the episodic versions include time-based resets, which we omit in the continuing testbeds. In these Atari environments, the agent has multiple lives, and the environment is reset when all lives are lost. Upon losing a life, a reward of -1 is issued as a penalty. Furthermore, in existing algorithmic solutions to episodic Atari testbeds, the rewards are transformed into $-1, 0$, or 1 by taking their sign for stable learning, though performance is evaluated based on the original rewards. We treat the transformed rewards as the actual rewards in our continuing testbeds, removing such inconsistency.

For each testbed-algorithm pair, we performed ten runs, and each run consisted of 3M steps for Mujoco testbeds and 5M steps for Atari testbeds. The learning curves corresponding to the best parameter setting for Mujoco and Atari testbeds are shown in Figure 1 (middle row) and Figure 3, respectively. The results show that SAC and DQN consistently perform the best in Mujoco testbeds and Atari testbeds, respectively.

	Task	DDPG		TD3		SAC		PPO	
		episodic	continuing	episodic	continuing	episodic	continuing	episodic	continuing
Reward rate	HalfCheetah	**13.48 ± 0.15**	**12.19 ± 1.41**	**9.72 ± 0.57**	**10.48 ± 1.69**	**11.64 ± 1.67**	**14.23 ± 0.77**	**3.57 ± 0.57**	**3.04 ± 0.74**
	Ant	-0.85 ± 0.30	**6.79 ± 0.37**	4.74 ± 0.26	**6.78 ± 0.09**	5.13 ± 0.82	**7.58 ± 0.20**	**4.48 ± 0.29**	**3.61 ± 0.47**
	Hopper	3.60 ± 0.05	**4.05 ± 0.06**	3.77 ± 0.05	**4.07 ± 0.04**	3.93 ± 0.05	**4.19 ± 0.07**	3.83 ± 0.07	**4.02 ± 0.07**
	Humanoid	**5.55 ± 0.19**	**6.50 ± 0.60**	5.83 ± 0.11	**7.75 ± 0.44**	6.34 ± 0.07	**8.09 ± 0.09**	5.25 ± 0.03	**7.65 ± 0.08**
	Walker2d	3.72 ± 0.17	**4.88 ± 0.19**	**4.82 ± 0.20**	**4.37 ± 0.50**	**3.05 ± 0.88**	**4.06 ± 0.83**	**5.23 ± 0.22**	**4.87 ± 0.29**
Number of resets	HalfCheetah	**0.50 ± 0.31**	**1.80 ± 0.96**	**0.30 ± 0.30**	**7.50 ± 5.67**	**1.20 ± 0.44**	**0.70 ± 0.30**	**0.20 ± 0.13**	**0.40 ± 0.31**
	Ant	**18.90 ± 8.49**	**23.00 ± 2.67**	**2.50 ± 0.87**	**1.20 ± 0.29**	**2.60 ± 0.99**	**5.70 ± 2.95**	**5.80 ± 1.16**	**4.50 ± 1.52**
	Hopper	**27.20 ± 1.68**	45.50 ± 1.92	**3.40 ± 1.90**	45.90 ± 1.60	**11.10 ± 2.25**	46.90 ± 2.42	**16.90 ± 2.52**	52.90 ± 1.88
	Humanoid	**80.70 ± 59.83**	**228.10 ± 75.23**	**0.10 ± 0.10**	55.30 ± 20.32	**1.00 ± 0.42**	5.50 ± 1.93	**61.70 ± 3.94**	107.40 ± 3.96
	Walker2d	**30.80 ± 2.44**	**42.50 ± 11.94**	**3.30 ± 1.04**	35.30 ± 15.76	**89.30 ± 32.12**	**103.70 ± 69.34**	**5.20 ± 0.70**	28.70 ± 6.15

Table 2: A comparison of the policy learned in the continuing task vs the policy learned in the corresponding episodic task. The upper group shows the mean and the standard error of the reward rates when deploying the learned policies obtained in these two settings for $10,000$ steps. The higher reward rate is marked in boldface, and the number obtained in other settings is also marked in bold if the difference is statistically insignificant. The lower group shows the number of resets within the evaluation steps. The reset number for the fewer is marked in boldface. This table shows that policies learned in continuing tasks make more frequent resets and achieve a higher reward rate.

As mentioned earlier, when resets are predefined, the agent may choose to solve a continuing or episodic task. We now illustrate the difference between these two choices by showing the difference between policies learned in these two tasks. The episodic tasks are the same as the above continuing tasks, except that the agent optimizes cumulative rewards only up to resetting. Table 2 shows the final reward rate and the number of resets when running in the continuing tasks for $10,000$ steps, the policies learned in the continuing and episodic Mujoco tasks. The results for Atari tasks demonstrate a similar trend as in Mujoco tasks and are shown in Table 13 (Appendix B).

Table 2 demonstrates that in most cases, learned policies in continuing tasks result in higher reward rates and more resets. This likely occurs because the reset cost is relatively small compared to the additional rewards gained through aggressive actions, which have a higher likelihood of causing resets. A follow-up experiment revealed that when a large reset cost is used, fewer resets are observed in most cases, and the reward rate, surprisingly, remains comparable in most instances and even higher in some, as shown in Table 3. This suggests that reset cost functions not only as a problem parameter but also as a solution parameter that requires tuning when applying current algorithms. Future research is needed to understand how to select this solution parameter.

	Task	DDPG		TD3		SAC		PPO	
	Reset cost	1	100	1	100	1	100	1	100
Reward rate (excluding reset cost)	HalfCheetah	**11.30 ± 1.35**	**10.46 ± 0.27**	**8.04 ± 1.79**	**6.15 ± 1.22**	**15.26 ± 0.30**	**13.28 ± 1.47**	**3.95 ± 0.39**	**3.83 ± 0.44**
	Ant	**4.26 ± 0.07**	3.34 ± 0.16	**2.02 ± 0.23**	**2.39 ± 0.23**	**7.26 ± 0.14**	**6.30 ± 0.60**	2.82 ± 0.44	**4.94 ± 0.15**
	Hopper	**2.85 ± 0.03**	**2.86 ± 0.04**	2.75 ± 0.05	**2.88 ± 0.04**	3.93 ± 0.13	**4.30 ± 0.04**	**3.96 ± 0.08**	**4.06 ± 0.08**
	Humanoid	6.88 ± 0.31	**8.02 ± 0.37**	6.96 ± 0.45	**8.02 ± 0.19**	**7.91 ± 0.19**	**7.51 ± 0.27**	**7.63 ± 0.08**	6.12 ± 0.06
	Walker2d	**3.79 ± 0.14**	**3.95 ± 0.11**	**2.64 ± 0.38**	**2.80 ± 0.46**	**4.70 ± 0.87**	**5.79 ± 0.19**	**5.10 ± 0.22**	**5.23 ± 0.18**
Number of resets	HalfCheetah	**2.20 ± 1.48**	**1.00 ± 0.33**	**8.10 ± 5.10**	**2.80 ± 1.91**	**0.20 ± 0.13**	**0.40 ± 0.16**	**36.30 ± 29.99**	**1.30 ± 0.47**
	Ant	94.20 ± 5.98	**65.20 ± 4.76**	89.80 ± 10.27	**58.40 ± 9.65**	**2.80 ± 1.17**	**4.80 ± 4.37**	80.50 ± 28.45	**4.80 ± 1.10**
	Hopper	84.30 ± 1.83	**69.70 ± 1.74**	100.20 ± 4.98	**86.60 ± 3.82**	57.30 ± 5.58	**35.80 ± 1.14**	53.40 ± 1.54	**44.00 ± 2.01**
	Humanoid	**161.90 ± 52.15**	**76.40 ± 46.33**	138.00 ± 38.23	**3.40 ± 1.82**	44.00 ± 15.10	**2.67 ± 1.50**	118.30 ± 4.72	**83.10 ± 3.52**
	Walker2d	104.50 ± 17.33	**39.70 ± 3.98**	108.90 ± 24.23	**55.40 ± 11.98**	**99.20 ± 71.06**	**3.00 ± 1.14**	27.70 ± 7.31	**10.70 ± 1.24**

Table 3: The table presents the reward rate and number of resets of the learned policies over 10,000 evaluation steps with varying reset costs. To ensure a fair comparison, the reset cost is excluded from the reward rate computation. The lower section of the table shows the number of resets during evaluation. The boldface represents the same meaning as in Table 2. These results demonstrate that policies learned in tasks with higher reset costs generally lead to fewer resets. In several cases (e.g., DDPG in Humanoid), higher reset costs are also associated with higher reward rates.

2.3 TESTBEDS WHERE THE AGENT CONTROLS RESETS

This section studies the behavior of current algorithms in continuing tasks where predefined resets are not available, and the agent decides when to reset. Intuitively, allowing the agent to choose when to reset can lead to higher reward rates compared to predefined resets, as the agent can optimize its behavior by avoiding unnecessary resets. However, predefined resets reduce the state and action spaces, making the testbeds easier. For instance, in environments like Humanoid, Walker, and Hopper, the agent needs to carefully control its actions to avoid falling, and recovering from these fallen states is difficult or impossible. In such cases, the agent must learn to recognize when it cannot recover and needs to reset the environment to continue. Predefined resets simplify the problem by eliminating these bad, unrecoverable states, allowing the agent to focus on learning in good states.

The testbeds are the five Mujoco testbeds used in Section 2.2 without predefined resets. In these new testbeds, the agent can choose to reset the environment at any time step. This is achieved by augmenting the environment's action space in these testbeds by adding one more dimension. This additional dimension has a range of $[0, 1]$, representing the probability of reset. The tested continuous control algorithms can then be readily applied, except that the exploration noise for this additional dimension needs to be set differently from other action dimensions because the performance of the policy is more sensitive to this dimension than the rest. We leave the details of the tested noises in Section A.3. The number of runs and number of steps in each run are chosen in the same way as in the above two subsections. The tested hyperparameters are provided in Section A.2. The learning curves, which are chosen the same way as the previous two subsections, are reported in Figure 1 (lower row) (Appendix B). We also show in Table 14 (Appendix B) the reward rate and the number of resets achieved by the final learned policy deployed for $10,000$ steps and compare it to the reward rates when the policies are learned in the testbeds with predefined resets.

Comparing the performance of the tested algorithms in testbeds with predefined resets and those with agent-controlled resets reveals some nuanced results. In many cases, algorithms trained in testbeds with agent-controlled resets achieved a similar final reward rate to those with predefined resets. In a few instances, algorithms in testbeds with agent-controlled resets performed better, achieving both higher final reward rates and more stable learning (e.g., PPO in HalfCheetah and Ant). Conversely, in other cases, the learned policies performed worse. Notably, some learning curves show a significant upward trend toward the end of training, suggesting that the performance differences may be due, at least in part, to the larger state and action spaces in the testbeds with agent-controlled resets, which could require more training time to fully optimize. Nevertheless, longer training time does not always suffice. For instance, in the Humanoid task, all algorithms performed considerably worse when resets were learned. The learning curves for most algorithms, except PPO, demonstrate slow improvement over time. DDPG faced such challenges that its final learned policy was even worse than the performance of a random policy in the Humanoid task with predefined resets (approximately 4.6). The failure in Humanoid likely stems from the fact that it has a significantly larger state space compared to other testbeds.

2.4 Failure to address large discount factors or offsets in rewards

Using the Mujoco testbeds presented above, we show in this section that the performance of all of the tested continuous control algorithms deteriorates significantly when a large discount factor is used or when all rewards are shifted by the large constant.

		Discount factor 0.99 → 0.999				All rewards +100			
	Algorithm	DDPG	TD3	SAC	PPO	DDPG	TD3	SAC	PPO
No resets	Swimmer	-85.95	-45.19	-99.23	46.84	-104.86	-103.20	-108.30	-101.52
	HumanoidStandup	-9.09	14.16	-60.13	-13.45	-29.16	5.70	-24.10	-11.97
	Reacher	-707.42	-6.01	-10.13	1.60	-429.94	-160.87	-117.87	-8.67
	Pusher	-13.80	-10.82	-7.23	-4.54	-183.44	-162.26	-25.07	-19.53
	SpecialAnt	-38.39	-67.71	-152.16	-11.86	-100.50	-44.65	-12.73	-42.30
Predefined resets	HalfCheetah	-20.62	49.84	4.26	-41.32	-59.69	-85.34	-44.86	-62.95
	Ant	-7.48	-22.46	-14.66	-15.50	-118.93	-97.01	-75.70	-31.90
	Hopper	-12.81	-8.45	-11.72	-21.73	-62.35	-53.05	-17.77	-36.00
	Humanoid	-34.28	-64.27	-74.81	-58.51	-83.17	-113.79	-109.34	-50.57
	Walker2d	-5.07	-15.12	-3.38	-29.89	-63.41	-53.33	-40.23	-52.50
Agent-controlled resets	HalfCheetah	-27.19	-29.79	-33.93	-26.41	-73.96	-26.31	-59.21	-78.05
	Ant	-5.32	10.74	-22.89	-19.04	-127.31	-85.00	-82.81	-69.68
	Hopper	-29.62	-11.62	-5.92	-12.87	-106.48	-65.56	-11.30	-36.35
	Humanoid	-155.59	2.13	-14.77	-23.05	-106.27	-108.54	-35.26	-21.05
	Walker2d	-30.49	6.14	-37.61	-22.96	-59.64	-46.72	-35.39	-77.86

Table 4: A large discount factor or reward offset hurt all tested algorithms' performance.

We report the percentage of improvement for each testbed-algorithm pair, defined as $\frac{\bar{r}^{0.999} - \bar{r}^{\text{random}}}{\bar{r}^{0.99} - \bar{r}^{\text{random}}} - 1$, where $\bar{r}^{0.999}$ is the final average reward rate over the last 10,000 steps with a discount factor of 0.999, and $\bar{r}^{0.99}$ is the reward rate with a discount factor of 0.99. The term $\bar{r}^{\text{random}}$ refers to the reward rate of a uniformly random policy. As in the previous subsections, all reward rates are averaged over ten runs, each of which has 3 million steps, and gray shading indicates that the difference between $\bar{r}^{0.99}$ and $\bar{r}^{0.999}$ is not statistically significant, as determined by Welch's t-test with a p-value less than 0.05. Additionally, we tested these pairs when all environment rewards were shifted by +/-100, with other experiment details the same as above. We report the percentage of improvement computed in a similar way as for discount factors when all environment rewards are shifted by +100 but with the common offset subtracted for a fair comparison. Formally, this percentage of improvement is $\frac{\bar{r}^{100} - 100 - \bar{r}^{\text{random}}}{\bar{r} - \bar{r}^{\text{random}}} - 1$, where $\bar{r}^{100}$ is the final average reward rate over the last 10,000 steps when all rewards are shifted by +100, and $\bar{r}$ is the reward rate without reward shifting. The results when all rewards are subtracted by -100 are similar and are thus omitted. The results (Table 4) show that, overall, algorithms with a discount factor of 0.999 perform much worse than those with 0.99. Moreover, a large reward offset leads to catastrophic failure across almost all task-algorithm pairs.

3 Evaluating Algorithms with Reward Centering

This section empirically shows that the temporal-difference-based reward centering method, originally introduced by Naik et al. (2024), improves or maintains the performance of all tested algorithms in the testbeds introduced in the previous section. Further, this method mitigates the negative effect when using a large discount factor and completely removes the detrimental effect caused by a large common reward offset.

The idea of reward centering stems from the following observation. By Laurent series expansion (Puterman, 2014), if a policy π results in a Markov chain with a single recurrent class, its discounted value function v_π can be decomposed into two parts, a state-independent offset $d_\pi^\top v_\pi = r(\pi)/(1-\gamma)$, where d_π is the stationary distribution under π, $r(\pi)$ is the average reward rate under policy π, and a state-dependent part keeping the relative differences among states. Here, the reward rate does not depend on the initial state due to the assumption of a single recurrent class. Note that only the state-dependent part is useful for improving the policy π. However, when the state-independent part has a large magnitude, possibly due to large offsets in rewards or a discount factor that is close to 1, approximating the state-independent part separately for each state can result in approximation errors that mask the useful state-dependent part.

Reward centering approximates the state-independent part using a shared scalar. Specifically, reward centering approximates a new discounted value function, obtained by subtracting all rewards by an approximation of $r(\pi)$, and this new discounted value function has a zero state-independent offset if the approximation of $r(\pi)$ is accurate. Even if the approximation of $r(\pi)$ is not accurate, removing a portion of the state-independent offset still helps.

	Task	DDPG	TD3	SAC	PPO
No resets	Swimmer	109.11	90.71	**1149.26**	**71.14**
	HumanoidStandup	**41.67**	19.79	**35.83**	**19.39**
	Reacher	-0.03	0.07	-0.11	**1.17**
	Pusher	**10.87**	**1.24**	0.39	**3.72**
	SpecialAnt	**12.67**	2.05	5.59	**10.55**
Predefined resets	HalfCheetah	**3.15**	13.13	5.05	4.66
	Ant	**22.22**	18.25	6.75	13.36
	Hopper	**2.53**	**14.83**	**4.44**	**4.56**
	Humanoid	**210.54**	**89.68**	**77.54**	**11.29**
	Walker2d	**16.28**	**10.37**	**7.72**	**7.37**
Agent-controlled resets	HalfCheetah	2.21	**17.45**	-2.05	4.06
	Ant	**12.73**	**94.13**	**34.57**	8.07
	Hopper	**42.28**	**15.72**	4.60	5.57
	Humanoid	246.73	**20.71**	**5.46**	**2.36**
	Walker2d	10.23	12.92	0.89	4.61
	Average improvement	49.54	28.07	89.06	11.46

Table 5: Percentage of reward rate improvement when applying reward centering to the tested algorithms in Mujoco testbeds.

A straightforward way to perform reward centering is to estimate $r(\pi)$ using an exponential moving average of all observed rewards. For on-policy algorithms, this moving average approach can guarantee convergence to $r(\pi)$. However, for off-policy algorithms, this approach does not converge to $r(\pi)$ (e.g., the behavior policy is uniformly random while the target policy is deterministic).

We now briefly describe the TD-based reward-centering approach, which can be applied to both on- and off-policy algorithms. This approach extends an approach to solve the average-reward criterion (Wan et al., 2021) to the discounted setting. Here, we illustrate this approach using use TD(0) (Sutton, 2018, p. 120), the simplest TD algorithm, as an example. More details on how tested algorithms employ this approach are provided in Section A.4.

Given transitions (S, R, S') generated by following some policy π, TD(0) estimates v_π by maintaining a table of value estimates $V : \mathbb{R}^{|\mathcal{S}|}$ and updating them using $V(S) \leftarrow V(S) + \alpha\delta$, where $\delta \overset{\text{def}}{=} R + \gamma V(S') - V(S)$ is a TD error, α is a step-size parameter, and γ is a discount factor. The TD-based reward-centering approach simply replaces the above TD error in TD(0) with the following new TD error:

$$\delta^{\text{RC}} \overset{\text{def}}{=} R - \bar{R} + \gamma V(S') - V(S),$$

where $\bar{R}$, a biased estimate of the reward rate, is also updated by the TD error δ^{RC}, as follows:

$$\bar{R} \leftarrow \bar{R} + \eta\alpha\delta^{\text{RC}},$$

where $\eta > 0$ is a constant. It is straightforward to show that, under certain asynchronous stochastic approximation assumptions on α, $V(s)$ converges to $v_\pi(s) - \frac{\eta}{1-\gamma+\eta|\mathcal{S}|}\sum_{s\in\mathcal{S}} v_\pi(s)$, following the same steps as in the proof of Theorem 1 by Naik et al. (2024). This result implies that although TD-based reward centering does not fully remove the state-independent offset $d_\pi^\top v_\pi$, it can remove a significant portion of it. Empirically, we also observed this effect.

Task	DQN	SAC	PPO
Breakout	-7.48	**1.67**	**11.51**
Pong	0.50	**51.94**	79.18
SpaceInvader	**20.97**	0.29	**19.72**
BeamRider	**7.01**	**35.00**	**75.67**
Seaquest	**26.79**	**22.75**	5.77
MsPacman	**9.96**	**1.76**	2.67
Average improvement	9.63	18.90	32.42

Table 6: Percentage of reward rate improvement when applying reward centering to the tested algorithms in Atari tasks. Statistically significant improvement percentage numbers are marked in boldface.

We evaluated algorithms with TD-based reward centering across all testbeds, comparing them to base algorithms that do not use reward centering. Each experiment was repeated ten times with different seeds, lasting 1 million steps for Mujoco testbeds and 5 million steps for Atari testbeds. We report the percentage improvement when using reward centering. Specifically, the reported number is $\frac{\bar{r}^{\text{RC}}-\bar{r}^{\text{random}}}{\bar{r}-\bar{r}^{\text{random}}} - 1$, where $\bar{r}^{\text{RC}}$ is the average of all received rewards, averaged across ten runs, with TD-based reward centering, $\bar{r}$ is defined similarly but without reward centering, and $\bar{r}^{\text{random}}$ is average-reward rate of a uniformly random policy. The reported value is the best result across all tested hyperparameter settings for both reward-centered and baseline algorithms. Shaded values indicate that performance differences are not statistically significant, according to Welch's t-test with $p < 0.05$. The reported results for Mujoco and Atari testbeds are shown in Table 5 and Table 6, respectively. The corresponding learning curves are provided in Appendix C. These results show that reward centering improves or maintains the performance of all of the tested algorithms in all testbeds. How much the performance improvement seems to depend on both the algorithm and the task.

In Tables 15 and 16, we show, using the Mujoco testbeds, that TD-based reward centering is most effective when using a large discount factor or when there is a large offset in rewards, echoing the findings by Naik et al. (2024) about DQN in smaller scale testbeds. However, unlike Naik et al.'s (2024) results, our results show that while the negative effect of large discount factors is much smaller with reward centering, it can still lead to notably worse performance in many cases. This suggests that when applying tested algorithms to solve complex continuing tasks, tuning the discount factor may still be valuable, even with reward centering.

We also evaluated the exponential moving average approach to perform reward centering. In addition, we evaluated another reward-centering approach inspired by Devraj and Meyn's (2021) relative Q-learning algorithms. The details of these two approaches are provided in Section A.4. The results in Tables 17 and 18 show that the moving-average-based approach works surprisingly well despite its theoretical unsoundness in off-policy algorithms, the reference-state-based approach helps in some cases while hurts the performance in some others, and the TD-based approach is the more effective than the other two.

4 Conclusions and Limitations

This paper empirically examines the challenges that continuing tasks with various reset scenarios pose to several well-known deep RL algorithms, using a suite of testbeds based on Mujoco and Atari environments. Our findings highlight key issues that future algorithmic advancements for continuing tasks may focus on. For instance, we demonstrate that the performance of tested algorithms can heavily depend on the availability of predefined resets, as these resets help agents escape traps and reduce the state space complexity. When predefined resets are available, all algorithms perform reasonably well, learning policies that exploit frequent resetting to achieve higher rewards. The reset cost balances this trade-off and also functions as a tuning parameter. In contrast, agent-controlled reset tasks are generally more challenging, and in some testbeds, allowing the agent to control resets significantly worsens performance. Additionally, we show that both a large discount factor and a large common offset in rewards can negatively impact the performance of all tested algorithms. Our results also validate the effectiveness of an existing approach to address these issues, demonstrating through extensive experiments that the negative impact of reward offset can be completely eliminated, while the harm from a large discount factor can be largely mitigated with a TD-based reward-centering approach. Even in scenarios with a smaller discount factor and no reward offset, this approach shows benefits across many testbeds for all tested algorithms.

This paper has several limitations. First, this paper focuses exclusively on the performance of online RL algorithms, leaving research on offline RL algorithms in continuing tasks unexplored. Second, although we concentrate on well-known discounted algorithms, it is worth investigating whether average-reward algorithms, such as those mentioned in Section 1, face similar challenges. Third, while most of the hyperparameters used in the experiments are standard choices and have been effective in episodic testbeds, they may not be ideal for continuing tasks. Identifying hyperparameter choices that are more suitable for continuing tasks remains unexplored. Despite these limitations, we believe that our findings provide valuable insights into the challenges of continuing tasks in deep RL, and they serve as a basis for future research.

REFERENCES

Abounadi, J., Bertsekas, D., and Borkar, V. S. (2001). Learning algorithms for Markov decision processes with average cost. *SIAM Journal on Control and Optimization*, 40(3):681–698.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Devraj, A. M. and Meyn, S. P. (2021). Q-learning with uniformly bounded variance. *IEEE Transactions on Automatic Control*, 67(11):5948–5963.

Eysenbach, B., Gu, S., Ibarz, J., and Levine, S. (2017). Leave no trace: Learning to reset for safe and autonomous reinforcement learning. *arXiv preprint arXiv:1711.06782*.

Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR.

Grand-Clément, J. and Petrik, M. (2024). Reducing blackwell and average optimality to discounted mdps via the blackwell discount factor. *Advances in Neural Information Processing Systems*, 36.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. (2018). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

Hisaki, Y. and Ono, I. (2024). Rvi-sac: Average reward off-policy deep reinforcement learning. *arXiv preprint arXiv:2408.01972*.

Lillicrap, T. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Ma, X., Tang, X., Xia, L., Yang, J., and Zhao, Q. (2021). Average-reward reinforcement learning with trust region methods. *arXiv preprint arXiv:2106.03442*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Naik, A., Wan, Y., Tomar, M., and Sutton, R. S. (2024). Reward centering. *arXiv preprint arXiv:2405.09999*.

Platanios, E. A., Saparov, A., and Mitchell, T. (2020). Jelly bean world: A testbed for never-ending learning. *arXiv preprint arXiv:2002.06306*.

Puterman, M. L. (2014). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Saxena, N., Khastagir, S., Kolathaya, S., and Bhatnagar, S. (2023). Off-policy average reward actor-critic with deterministic policy search. In *International Conference on Machine Learning*, pages 30130–30203. PMLR.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sharma, A., Xu, K., Sardana, N., Gupta, A., Hausman, K., Levine, S., and Finn, C. (2021). Autonomous reinforcement learning: Formalism and benchmarking. In *International Conference on Learning Representations*.

Sharma, A., Xu, K., Sardana, N., Gupta, A., Hausman, K., Levine, S., and Finn, C. (2022). Autonomous reinforcement learning: Formalism and benchmarking. In *International Conference on Learning Representations*.

Sutton, R. S. (2018). Reinforcement learning: An introduction. *A Bradford Book*.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., et al. (2024). Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*.

Wan, Y., Naik, A., and Sutton, R. S. (2021). Learning and planning in average-reward Markov decision processes. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pages 10653–10662.

Zhang, Y. and Ross, K. W. (2021). On-policy deep reinforcement learning for the average-reward criterion. In *International Conference on Machine Learning*, pages 12535–12545. PMLR.

Zhao, R., Abbas, Z., Szepesvári, D., Naik, A., Holland, Z., Tanner, B., and White, A. (2022). Csuite: Continuing environments for reinforcement learning. *Github: google-deepmind/csuite*.

Zhu, H., Yu, J., Gupta, A., Shah, D., Hartikainen, K., Singh, A., Kumar, V., and Levine, S. (2020). The ingredients of real-world robotic reinforcement learning. *arXiv preprint arXiv:2004.12570*.

Zhu, Z., Braz, R. d. S., Bhandari, J., Jiang, D., Wan, Y., Efroni, Y., Wang, L., Xu, R., Guo, H., Nikulkov, A., et al. (2023). Pearl: A production-ready reinforcement learning agent. *arXiv preprint arXiv:2312.03814*.

A Details of Experiment Setup

This appendix provides details on the experiments conducted to produce the results presented in the main text and the subsequent two appendices. First, we present additional hyperparameters used by the algorithms tested in testbeds without resets or with predefined resets. Next, we describe the modifications made to the tested algorithms for testbeds with agent-controlled resets. Following this, we provide detailed information about how the tested algorithms are used together with reward centering. In the main text, we introduced the TD-based reward centering approach; here, we describe two additional approaches for performing reward centering, outlining how these methods were applied to the tested algorithms, along with the values of additional hyperparameters tested for reward centering.

A.1 Average-reward rate as the evaluation metric

In reinforcement learning (RL), an agent interacts with an environment to learn how to make decisions that maximize a cumulative reward signal. The environment is typically modeled as a finite Markov Decision Process (MDP), which consists of a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p)$, where $\mathcal{S}$ represents the set of states, $\mathcal{A}$ the set of actions, $\mathcal{R}$ is the set of rewards, $p(s', r \mid s, a)$ is the probability of transitioning from state s to s' and observing a reward of r, given action a. At each time step t, the agent observes the current state S_t, selects an action A_t based on a policy, and receives a reward signal R_{t+1} from the environment, with the goal of learning a policy that maximizes long-term reward.

For continuing tasks, where the agent-environment interaction persists indefinitely, the average-reward criterion is suitable as the performance metric and is therefore used in this paper. Let the initial state be s_0, the average reward is defined as $r(\pi, s_0) \stackrel{\text{def}}{=} \lim_{T\to\infty} \frac{1}{T}\mathbb{E}\left[\sum_{t=1}^{T} R_t \middle| A_t \sim \pi(\cdot \mid S_t), S_0 = s_0\right]$, where $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ is the agent's policy.

While there are several deep RL algorithms (e.g., Zhang and Ross 2021) addressing the average-reward criterion, we choose to study several well-known discounted deep RL algorithms. This is because the focus of this paper is on the challenges of continuing tasks rather than on studying the properties of algorithms, and these discounted algorithms have been better understood in the literature. Further, note that by adjusting the discount factor to be close to one, discounted algorithms can approximately solve the average-reward criterion in continuing tasks. When the discount factor is sufficiently close to one, any discounted optimal policy is also average-reward optimal (Grand-Clément and Petrik, 2024).

A.2 Tested Hyperparameter for Algorithms in testbeds without resets or with predefined resets

We provide hyperparameters used by the tested algorithms in Tables 7—12.

Hyperparameter	Value
Actor & critic networks	fully connected with 256 $\times$ 256 hidden layers and Relu activation
Optimizer	Adam
Discount factor	0.99, 0.999
Actor & critic learning rates	3e-4
Actor & critic target smoothing coefficients	0.005
Batch size	256
Replay buffer size	1e6
Exploration noise distribution	Normal(0, 0.1)
Warmup stage (taking random actions)	first 25000 steps
Learning after	first 25000 steps

Table 7: Tested DDPG and TD3's hyperparameters for Mujoco testbeds. TD3, in addition, makes a delayed actor update every other critic update. The noise added in the sample action used in TD3's update is a zero mean Normal distribution with noise 0.2. This noised sampled action is then clipped to be within $[-0.5, 0.5]$.

Hyperparameter	Value
Actor & critic networks	fully connected with 256 $\times$ 256 hidden layers and Relu activation
Optimizer	Adam
Discount factor	0.99, 0.999
Actor learning rate	3e-4
Critic learning rate	1e-3
Use autotune	True
Critic target smoothing coefficient	0.005
Batch size	256
Replay buffer size	1e6
Warmup stage (taking random actions)	first 5000 steps
Learning after	first 5000 steps

Table 8: Tested SAC's hyperparameters for Mujoco testbeds

Hyperparameter	Value
Actor & critic networks	fully connected with 64 $\times$ 64 hidden layers and Tanh activation
Optimizer	Adam
Discount factor	0.99, 0.999
Actor & critic learning rates	3e-4
λ in generalized advantage estimation	0.95
Importance sampling ratio clipping range	[0.8, 1.2]
Samples collected for updates	2048
Batch size	64
Number of updates per sample	10
Gradient norm clipping threshold	0.5
Normalize advantage	True
Value clipping	False
Return normalization	False
Entropy coefficient	0.0

Table 9: Tested PPO's hyperparameters for Mujoco testbeds

Hyperparameter	Value
Q network	three convolution layers followed by one fully connected layer, all with Relu activation
Conv layer 1	kernel size 8, output channel size 32, strides 4, paddings 0
Conv layer 2	kernel size 4, output channel size 64, strides 2, paddings 0
Conv layer 3	kernel size 3, output channel size 64, strides 1, paddings 0
Fully connected layer size	512
Optimizer	Adam
Discount factor	0.99, 0.999
Learning rate	1e-4
Replay buffer size	800000
Samples between two updates	4
Target network update	every 1000 steps
Batch size	64
Number of updates per sample	10
Normalize advantage	True
Gradient norm clipping threshold	0.5
Exploration	ϵ-greedy with linear decay. ϵ starts from $\epsilon = 1$ and ends at 0.01. 1000000 decay steps.
Learning after	first 80000 steps

Table 10: Tested hyperparameters for DQN for Atari testbeds.

Hyperparameter	Value
Actor & critic networks	three convolution layers followed by one fully connected layer, all with Relu activation
Conv layer 1	kernel size 8, output channel size 32, strides 4, paddings 0
Conv layer 2	kernel size 4, output channel size 64, strides 2, paddings 0
Conv layer 3	kernel size 3, output channel size 64, strides 1, paddings 0
Fully connected layer size	512
Optimizer	Adam
Discount factor	0.99, 0.999
Actor & critic learning rate	3e-4
λ in generalized advantage estimation	0.95
Importance sampling ratio clipping range	[0.9, 1.1]
Samples collected for updates	1024
Batch size	256
Number of updates per sample	8
Gradient norm clipping threshold	0.5
Normalize advantage	True
Value clipping	False
Return normalization	False
Entropy coefficient	0.01

Table 11: Tested hyperparameters for PPO for Atari testbeds.

Hyperparameter	Value
Actor & critic networks	three convolution layers followed by one fully connected layer, all with Relu activation
Conv layer 1	kernel size 8, output channel size 32, strides: 4, paddings: 0
Conv layer 2	kernel size 4, output channel size 64, strides: 2, paddings: 0
Conv layer 3	kernel size 3, output channel size 64, strides: 1, paddings: 0
Fully connected layer size	512
Optimizer	Adam
Discount factor	0.99, 0.999
Actor & critic learning rates	3e-4
Use autotune	False
Entropy coefficient	0.2
Samples collected between two updates	4
Critic target update frequency	2000
Batch size	64
Replay buffer size	800000
Warmup stage (taking random actions)	first 20000 steps
Learning after	first 20000 steps

Table 12: Tested hyperparameters for SAC for Atari testbeds.

A.3 Hyperparameters when applied to testbeds with agent-controlled resets

We modified the hyperparameters of the tested algorithms in two ways to improve the algorithms' performance in testbeds with agent-controlled resets.

First, we adjust a hyperparameter that controls the level of exploration for DDPG, TD3, and SAC. For DDPG and TD3, the exploration noise is a sample of a zero-mean multivariate Gaussian random vector with independent elements. This exploration noise is then added to the action generated by the actor network to perform persistent exploration. For testbeds without resets or with predefined resets, we applied the same standard deviation of 0.1 to all elements. However, when resets are part of actions, we tested smaller standard deviations, including 0.05, 0.005, 0.0005, and 0.00005, for the reset dimension. This is because, compared to the other dimensions in actions, a small noise in the reset dimension would have a significant effect on the behavior of the policy. For SAC, the entropy regularization coefficient controls the level of exploration. We applied the autotune technique introduced by Haarnoja et al. (2018) to adjust this coefficient dynamically. This technique introduces some regularization that pushes the entropy of the learned policy toward some predefined target value, guaranteeing that exploration does not diminish to zero asymptotically. For testbeds without resets or with predefined resets, the target entropy was chosen to be $-|\mathcal{A}|$, a choice tested by Haarnoja et al. (2018), where $|\mathcal{A}|$ is the dimension of the action space. When resetting is part of the action, we found this choice leads to very frequent resets, even at the end of training. We therefore tested smaller target entropy values, including $-|\mathcal{A}|, -|\mathcal{A}| - 3, -|\mathcal{A}| - 6$, and $-|\mathcal{A}| - 9$. PPO's exploration noise is learned, and there is no mechanism for maintaining exploration above a certain level or pushing exploration toward a certain level. Therefore, no more changes need to be applied to PPO's hyperparameters.

The second change we made was to have a different random policy for collecting data in the warmup stage of DDPG, TD3, and SAC. In testbeds without resets or with predefined resets, a policy that uniformly randomly samples from the action space was used in the warmup stage. When resetting probability is part of the action, we apply a different policy that is biased toward lower reset probability. The reason is that a uniformly random policy would output a reset probability of 0.5, which is so high that most of the data collected following this policy will be several steps away from the initial states. To generate longer trajectories, we chose the resetting probability element of the action to be $1/N$, where N is an integer sampled uniformly from $1, 2, \ldots, 1000$, and kept other elements uniformly sampled.

A.4 Applying Reward Centering Methods to the Tested Algorithms

In this section, we describe how we applied three reward-centering approaches to the tested algorithms. We start with TD-based reward centering and then discuss two other alternative approaches.

We apply TD-based reward centering to DQN the same way as Naik et al. (2024) did. DQN maintains an approximate action-value function, $q_w : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, with the vector w being the parameters of the function. To update w, DQN maintains a target network $q_{\hat{w}}$ parameterized the same way as q_w but with different parameters. For every fixed number of time steps, the values of w are copied to $\hat{w}$. Every time step, DQN samples a batch of transition tuples $(s_i, a_i, r_i, s'_i), i \in \{1, 2, \ldots, n\}$ from the replay buffer, where s_i, a_i, r_i, s'_i denote a state, an action, and the resulting reward and state, respectively, and n is the batch size. The update rule to w is

$$w \stackrel{\text{def}}{=} w + \alpha \frac{1}{n} \sum_{i=1}^{n} \delta_i \nabla_w q_w(s_i, a_i), \tag{1}$$

where $\delta_i \stackrel{\text{def}}{=} r_i + \gamma \max_{a \in \mathcal{A}} q_{\hat{w}}(s'_i, a) - q_w(s_i, a_i)$ is a TD error, α is a step-size parameter and γ is a discount factor. With TD-based reward centering, we update w with equation 1 but with δ_i replaced by a different TD error, where the reward is subtracted by an offset $\bar{r}$, defined as follows:

$$\delta_i^{\text{RC}} \stackrel{\text{def}}{=} \delta_i - \bar{r}. \tag{2}$$

The offset $\bar{r}$ is updated whenever w is updated, using the new TD errors, following

$$\bar{r} \stackrel{\text{def}}{=} \bar{r} + \beta \frac{1}{n} \sum_{i=1}^{n} \delta_i^{\text{RC}}, \tag{3}$$

where β is another step-size parameter. The tested β values in our experiments are $3e-2, 1e-2, 3e-3, 1e-3, 3e-4$. The tested discount factors in our experiments are $0.99, 0.999$, and 1.0. These hyperparameters were also used in other tested algorithms with reward centering.

DDPG, TD3, SAC, and PPO are also driven by TD-learning with various different TD errors. We show their respective TD errors below. The centered versions of DDPG, TD3, and SAC can be derived straightforwardly by replacing their respective TD errors with the new TD errors obtained, as in equation 2, and updating $\bar{r}$ whenever their critic parameters are updated, as in equation 3. PPO requires a slightly more complicated treatment in the update of $\bar{r}$, which we will discuss separately.

Like DQN, DDPG also samples a batch of transition tuples $(s_i, a_i, r_i, s'_i), i \in \{1, 2, \ldots, n\}$ from the replay buffer to update the weight vector w. In addition, the algorithm samples an action a'_i according to the actor's policy for each s'_i. DDPG's TD error is $\delta_i \stackrel{\text{def}}{=} r_i + q_{\hat{w}}(s'_i, a'_i) - q_w(s_i, a_i)$.

TD3 maintains two approximate value functions that are parameterized in the same way but with different parameters. Denote them by q_{w_1}, q_{w_2}. To update w_1 and w_2, for each time step, just like DDPG, TD3 samples a batch of transition tuples $(s_i, a_i, r_i, s'_i), i \in \{1, 2, \ldots, n\}$ from the replay buffer and a batch of actions a'_i. Unlike in DDPG, an additional Gaussian noise ϵ_i is added on a'_i. TD3's TD error is $r_i + \gamma \left(\min_{j \in \{1,2\}} q_{w_j}(s'_i, a'_i + \epsilon_i)\right) - q_{w_j}(s_i, a_i)$.

SAC also employs two approximate value functions q_{w_1}, q_{w_2}. Let $(s_i, a_i, r_i, s'_i), i \in \{1, 2, \ldots, n\}$ and a'_i be generated the same way as in DDPG. The continuous control version of SAC's TD error is $r_i + \gamma \left(\min_{j \in \{1,2\}} q_{w_j}(s'_i, a'_i)\right) - \kappa \log \pi(a'_i \mid s'_i) - q_{w_j}(s_i, a_i)$, where κ is a regularization coefficient influencing the entropy of the policy and is either predefined or automatically tuned, and π is the actor's policy. The discrete control version of SAC does not use sampled actions a'_i but considers all possible actions and uses the expectation. Its TD error is $\delta_i \stackrel{\text{def}}{=} r_i + \gamma \left(\sum_{a \in \mathcal{A}} \pi(a \mid s'_i) \left(\min_{j \in \{1,2\}} q_{w_j}(s'_i, a)\right) - \kappa \log \pi(a \mid s'_i)\right) - q_{w_j}(s_i, a_i)$.

PPO does not maintain an approximate action-value function but an approximate state-value function $v_w : \mathcal{S} \to \mathbb{R}$, with w being the weight vector. PPO proceeds in rounds. For each round, PPO collects a certain number of transitions following the current policy without changing any parameters and then applies multiple updates to both actor and critic parameters using the transitions. These transitions are not used in subsequent rounds. Let S_t, A_t denote the state, action at time step t and let R_{t+1} denote the resulting reward. PPO's TD error at time step t is defined as follows:

$$\delta_t \stackrel{\text{def}}{=} R_{t+1} + \gamma v_w(S_{t+1}) - v_w(S_t).$$

From this TD error, generalized advantage estimate (GAE) and truncated λ-return are computed, which are used to update actor and critic parameters. The centered TD error for PPO is defined by $\delta_t^{\text{RC}} \stackrel{\text{def}}{=} \delta_t - \bar{r}$, as in equation 2.

However, unlike the above algorithms, the update to $\bar{r}$ is performed using all transitions collected in a round instead of a batch of transitions because this does not add too much additional computation, given that the TD errors for all transitions visited in the round need to be computed anyway, to obtain the GAE and truncated λ-return.

Regarding the update to $\bar{r}$, another difference between PPO and the above algorithms is that in PPO, $\bar{r}$ is not performed every time the critic is updated, but every time the TD error is computed, to save computation. Recall that PPO's parameter updates proceed in epochs. For each epoch, all transitions collected in the current round are used for one time in both actor and critic updates. Even if there are multiple critic updates within each epoch, the TD errors are only computed once at the beginning of each epoch.

The above discussion finishes with a discussion of how we apply TD-based reward centering to the tested algorithms. We now discuss the other two reward-centering approaches.

The first approach is to simply let $\bar{r}$ be updated with an exponential moving average of the past rewards instead of being updated by equation 3. Formally, at time step t, $\bar{r}$ is updated with

$$\bar{r} \leftarrow \beta \bar{r} + (1 - \beta) R_t,$$

where $\beta \in [0, 1]$ is the moving average rate. The tested β values are $0.99, 0.999, 0.9999$. As suggested by Naik et al. (2024), this approach is theoretically sound in on-policy algorithms, such as

PPO, but is problematic for off-policy algorithms, such as the rest of the tested algorithms. In our paper, we empirically test this approach for all algorithms.

The other approach uses a set of reference states and is based on the relative Q-learning family of algorithms proposed by Devraj and Meyn (2021). These algorithms are tabular discounted algorithms and can be viewed as the extension of relative-value-iteration(RVI)-based Q-learning (Abounadi et al., 2001), a family of average-reward algorithms, to the discounted setting. Here, we briefly discuss the idea of relative Q-learning. We will then mention how to perform reward centering in the tested algorithms following the same idea.

Relative Q-learning maintains a $\mathcal{S} \times \mathcal{A}$-sized table of estimates for action values and updates these estimates in a similar way as Q-learning. For each time step, a state S_t is observed, and an action A_t is chosen by a policy that may or may not be controlled by the agent; the algorithm then updates with the resulting transition $(S_t, A_t, R_{t+1}, S_{t+1})$ using the following update rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} - f(Q) + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t) \right), \tag{4}$$

where f is a function satisfying certain properties. Examples of such functions include $f(Q) = \frac{1}{|\mathcal{S}| \times |\mathcal{A}|} \sum_{s \in \mathcal{S}, a \in \mathcal{A}} Q(s,a)$, $f(Q) = \max_{s \in \mathcal{S}, a \in \mathcal{A}} Q(s,a)$, or $f(Q) = \min_{s \in \mathcal{S}, a \in \mathcal{A}} Q(s,a)$. Here $f(Q)$ is a common offset subtracted by all rewards, therefore serving the same role as $\bar{r}$.

We make a few observations regarding this algorithm. First, note that the f function is chosen before the agent starts and is fixed through the agent's lifetime. Second, note that, unlike $\bar{r}$, $f(Q)$ does not need to be estimated separately. Third, note that the centered Q-learning algorithm introduced by Naik et al. (2024) with tabular representation can be written in equation 4 with $f(Q) \stackrel{\text{def}}{=} \eta \sum_{s \in \mathcal{S}, a \in \mathcal{A}} Q(s,a)$, where η is a hyperparameter. However, the relation between the two algorithms with function approximation is unclear.

Following the above idea, we replaced $\bar{r}$ in the tested algorithms by $f(q_w) \stackrel{\text{def}}{=} \frac{1}{|\mathcal{I}|} \sum_{(s,a) \in \mathcal{I}} q_w(s,a)$, where $\mathcal{I}$ is a fixed set of state-action pairs or $f(q_{w_j}) \stackrel{\text{def}}{=} \frac{1}{2|\mathcal{I}|} \sum_{(s,a) \in \mathcal{I}} (q_{w_1}(s,a) + q_{w_1}(s,a))$ when two value functions are used. The size of $\mathcal{I}$ is the same as the batch size used in each algorithm. The pairs are sampled randomly from the replay buffer right before the first learning update. Readers may refer to Tables 7—12 for the first learning update time step.

B Additional Evaluation Results of Tested RL Algorithms

This appendix shows evaluation results of tested RL algorithms that are omitted in Section 2 of the main text.

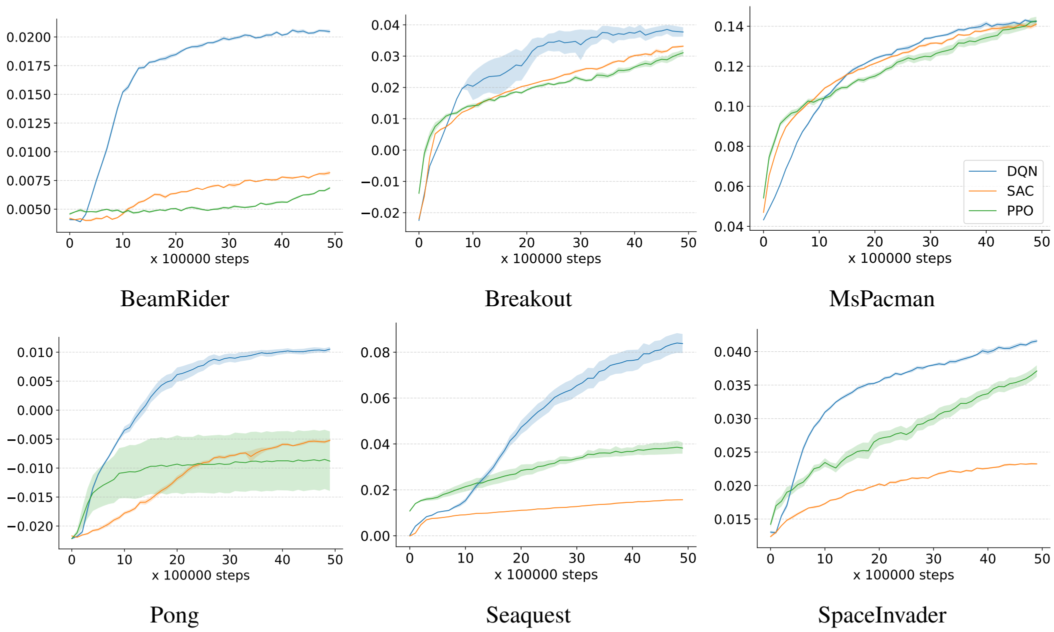

Figure 3: Learning curves in continuing testbeds with predefined resets based on the Atari environment. Each point shows the reward rate over the past 100k steps. Shading area standards for one standard error. Overall, DQN performs the best of the three tested algorithms.

		DQN		SAC		PPO	
		episodic	continuing	episodic	continuing	episodic	continuing
Reward rate	Breakout	**3.43 ± 0.36**	**4.01 ± 0.21**	2.08 ± 0.61	**3.51 ± 0.08**	**3.02 ± 0.07**	**3.05 ± 0.13**
	Pong	0.76 ± 0.07	**1.01 ± 0.07**	**-0.64 ± 0.16**	**-0.51 ± 0.04**	**-0.80 ± 0.51**	**-0.89 ± 0.52**
	SpaceInvader	3.87 ± 0.16	**4.23 ± 0.09**	2.17 ± 0.04	**2.36 ± 0.03**	3.40 ± 0.07	**3.73 ± 0.12**
	BeamRider	1.94 ± 0.05	**2.06 ± 0.03**	0.75 ± 0.04	**0.84 ± 0.02**	**0.65 ± 0.01**	**0.69 ± 0.02**
	Seaquest	5.01 ± 0.30	**8.50 ± 0.54**	0.31 ± 0.10	**1.57 ± 0.03**	3.13 ± 0.20	**3.89 ± 0.28**
	MsPacman	12.45 ± 0.15	**14.59 ± 0.17**	10.08 ± 0.25	**13.85 ± 0.19**	13.62 ± 0.28	**14.47 ± 0.23**
Num resets	Breakout	**39.80 ± 11.86**	**32.50 ± 2.18**	84.50 ± 26.40	**32.70 ± 0.68**	**35.10 ± 0.97**	52.30 ± 2.97
	Pong	**4.30 ± 0.21**	4.90 ± 0.18	**3.90 ± 0.53**	**4.50 ± 0.17**	**7.80 ± 0.81**	**7.90 ± 0.87**
	SpaceInvader	**12.70 ± 0.76**	19.30 ± 0.94	**10.30 ± 0.63**	32.30 ± 1.05	**36.60 ± 1.05**	**37.70 ± 0.86**
	BeamRider	**5.10 ± 0.87**	15.00 ± 1.41	**3.10 ± 0.59**	9.40 ± 0.75	**18.70 ± 0.76**	**20.10 ± 0.69**
	Seaquest	**10.60 ± 0.60**	13.70 ± 1.52	**3.30 ± 1.24**	38.00 ± 1.22	**18.20 ± 0.57**	**17.20 ± 0.20**
	MsPacman	**32.80 ± 0.55**	**34.90 ± 1.14**	**31.30 ± 0.70**	34.90 ± 0.53	**33.30 ± 0.78**	**34.70 ± 0.84**

Table 13: A comparison of policies learned in the continuing Atari testbeds versus policies learned in the corresponding episodic testbeds. The upper group shows the mean and the standard error of the reward rates when deploying the learned policy obtained in these two settings for $10,000$ steps. The higher reward rate is marked in boldface, and the number obtained in other settings is also marked in bold if the difference is statistically insignificant. The lower group shows the number of resets within the evaluation steps, with the fewer number of resets indicated in bold. This table shows that policies learned in continuing testbeds make more frequent resets and achieve a higher reward rate.

	Task	DDPG		TD3		SAC		PPO	
		agent-controlled	predefined	agent-controlled	predefined	agent-controlled	predefined	agent-controlled	predefined
Reward rate	HalfCheetah	**13.15 ± 0.20**	**12.19 ± 1.41**	**12.03 ± 0.43**	**10.48 ± 1.69**	12.69 ± 0.28	**14.23 ± 0.77**	**6.81 ± 0.49**	3.04 ± 0.74
	Ant	**7.32 ± 0.14**	**6.79 ± 0.37**	5.97 ± 0.25	**6.78 ± 0.09**	6.90 ± 0.14	**7.58 ± 0.20**	**4.41 ± 0.37**	**3.61 ± 0.47**
	Hopper	**4.09 ± 0.17**	**4.05 ± 0.06**	**4.34 ± 0.06**	4.07 ± 0.04	**4.05 ± 0.09**	**4.19 ± 0.07**	3.55 ± 0.16	**4.02 ± 0.07**
	Humanoid	**4.10 ± 1.30**	**6.50 ± 0.60**	5.38 ± 0.04	**7.75 ± 0.44**	5.88 ± 0.08	**8.09 ± 0.09**	6.82 ± 0.06	**7.65 ± 0.08**
	Walker2d	4.30 ± 0.14	**4.88 ± 0.19**	**4.16 ± 0.18**	**4.37 ± 0.50**	**4.29 ± 0.29**	**4.06 ± 0.83**	**4.56 ± 0.25**	**4.87 ± 0.29**
Num resets	HalfCheetah	**1.40 ± 0.34**	**1.80 ± 0.96**	**1.20 ± 0.25**	**7.50 ± 5.67**	6.50 ± 1.28	**0.70 ± 0.30**	6.00 ± 1.37	**0.40 ± 0.31**
	Ant	**7.10 ± 1.87**	23.00 ± 2.67	10.20 ± 1.17	**1.20 ± 0.29**	**2.30 ± 0.63**	**5.70 ± 2.95**	66.60 ± 24.96	**4.50 ± 1.52**
	Hopper	**37.30 ± 3.23**	45.50 ± 1.92	**41.00 ± 1.54**	45.90 ± 1.60	**47.70 ± 1.63**	**46.90 ± 2.42**	**56.50 ± 3.37**	**52.90 ± 1.88**
	Humanoid	**1102.90 ± 988.81**	**228.10 ± 75.23**	255.10 ± 7.46	**55.30 ± 20.32**	166.80 ± 14.14	**5.50 ± 1.93**	125.00 ± 5.95	**107.40 ± 3.96**
	Walker2d	74.70 ± 6.92	**42.50 ± 11.94**	**64.30 ± 26.00**	**35.30 ± 15.76**	**91.00 ± 25.29**	**103.70 ± 69.34**	**31.70 ± 5.66**	**28.70 ± 6.15**

Table 14: A comparison of policies learned in testbeds with predefined resets versus those learned in testbeds with agent-controlled resets. The upper group shows the mean and the standard error of the reward rates when deploying learned policies obtained in these two settings for $10,000$ steps. The higher reward rate is highlighted in bold, and if the difference is statistically insignificant, both values are also marked in bold. The lower group shows the number of resets within the evaluation steps, with the fewer number of resets indicated in bold. In general, algorithms achieve a higher reward rate and lower reset frequency when running on testbeds with predefined resets compared to those where resets are controlled by the agent.

C ADDITIONAL RESULTS OF ALGORITHMS WITH REWARD CENTERING

This appendix presents results concerning reward centering that are omitted in the main text. We will start with the result showing the usefulness of TD-based reward centering when a large discount factor or a large reward offset is present. We will then compare the three reward-centering approaches detailed in Section A.4.

To examine the influence of the discount factor with TD-based reward centering, we show the percentage of improvement of the asymptotic reward rate when using a discount factor of 0.999 and 1.0, as compared to when using the discount factor of 0.99, for centered algorithms in Mujoco testbed. Section 2.4 shows the formal definition of this percentage of improvement. As a baseline, we show the percentage of improvement when using a discount factor of 0.999, as compared to a discount factor of 0.99, for the based algorithms (c.f. Table 4). Discount factor 1.0 is not used by the base algorithms because approximate values can diverge to infinity. The results shown in Table 15 suggest that centered algorithms are indeed significantly less sensitive to the choice of the discount factor when resets are available. In several testbeds without resets, including Swimmer, HumanoidStandup, and SpecialAnt, increasing the discount factor can hurt performance significantly. This is potentially due to the fact that none of the tested algorithms, regardless of whether they use reward centering or not, successfully solve these testbeds even with a discount factor of 0.99. A larger discount factor increases the difficulty by optimizing a longer-term value, resulting in even worse performance of the tested algorithms.

To examine the influence of reward offsets, we show the percentage of improvement of the asymptotic reward rate when shifting all rewards by -100 or +100 for both centered and base algorithms in the Mujoco testbeds. Section 2.4 shows the formal definition of this percentage of improvement. The results (Table 16) show that centered algorithms are not sensitive to reward offsets at all, while uncentered algorithms are extremely sensitive to the offsets.

Overall, our experiments confirm the effectiveness of TD-based reward centering by showing that it can be combined with all tested algorithms and improve their performance. Further, centered algorithms work well with a large discount factor, especially for testbeds with resets, making the selection of an appropriate discount factor easier. Finally, the centered algorithm is not sensitive to reward offsets at all.

	Algorithm	DDPG			TD3			SAC			PPO		
	Use TD-based RC	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
	Discount factor	0.999	1.0	0.999	0.999	1.0	0.999	0.999	1.0	0.999	0.999	1.0	0.999
No resets	Swimmer	**-88.54**	**-96.33**	**-85.95**	71.53	75.63	-45.19	-52.03	**-95.04**	-99.23	**102.98**	**122.57**	46.84
	HumanoidStandup	-11.37	**-38.71**	-9.09	-5.49	-11.43	14.16	**-46.96**	**-52.99**	**-60.13**	-9.10	**-13.72**	-13.45
	Reacher	**-62.35**	**-56.17**	**-707.42**	-1.88	-0.82	**-6.01**	-1.28	0.48	-10.13	-1.35	-1.42	1.60
	Pusher	-3.79	-4.05	**-13.80**	-2.83	-3.34	**-10.82**	-2.93	**-2.78**	**-7.23**	**-3.39**	**-4.06**	**-4.54**
	SpecialAnt	32.19	-9.86	**-38.39**	**-38.02**	**-56.02**	**-67.71**	**-153.05**	**-148.25**	**-152.16**	-6.75	**-23.00**	-11.86
Predefined resets	HalfCheetah	-10.82	**15.43**	-20.62	20.16	15.41	49.84	-6.24	-6.06	4.26	7.59	**-19.77**	**-41.32**
	Ant	-2.84	-0.57	**-7.48**	-2.31	**-2.98**	**-22.46**	-2.50	0.75	**-14.66**	9.33	-6.29	-15.50
	Hopper	-2.18	-2.07	-12.81	0.93	0.53	**-8.45**	-0.65	**-6.78**	**-11.72**	-1.46	-6.20	**-21.73**
	Humanoid	-8.19	-1.58	-34.28	**-29.44**	**-30.60**	**-64.27**	-14.40	**-14.99**	**-74.81**	-6.10	-2.34	**-58.51**
	Walker2d	0.25	1.35	-5.07	**-15.93**	**-13.34**	**-15.12**	-4.25	-9.44	-3.38	-5.25	-10.15	**-29.89**
Agent-controlled resets	HalfCheetah	2.28	-8.50	**-27.19**	-4.49	-4.07	**-29.79**	-6.20	-2.15	**-33.93**	1.10	9.42	**-26.41**
	Ant	1.06	-1.55	**-5.32**	4.15	2.95	10.74	-4.76	**-20.34**	-22.89	-11.63	-6.85	**-19.04**
	Hopper	-9.09	**-26.14**	**-29.62**	0.54	0.47	**-11.62**	2.85	3.84	**-5.92**	-0.29	-2.23	**-12.87**
	Humanoid	-37.50	**-86.99**	**-155.59**	3.60	-0.17	2.13	-1.66	-3.29	**-14.77**	-1.53	-3.27	**-23.05**
	Walker2d	**-17.47**	**-23.27**	**-30.49**	-11.93	-7.72	6.14	-5.48	**-22.19**	**-37.61**	**-13.20**	-12.19	**-22.96**

Table 15: TD-based reward centering is less sensitive to the choice of the discount factor than noncentered methods

We now compare the three reward-centering methods. Tables 17 and 18 display the percentage of improvement for each method, with results for the TD-based approach drawn from Tables 5 and 6. The experimental setup and method for calculating the percentage improvement are detailed in Section 3, while hyperparameters specific to each reward-centering method are provided in Section A.4. We show the learning curves of the base algorithms and the algorithms with various reward-centering approaches in Figures 4—7.

The results indicate that the TD-based approach performs best among the three methods tested. Interestingly, the moving-average approach also performed well in off-policy algorithms, which was unexpected. The reference-state-based approach showed mixed results, improving performance in some cases but diminishing it in others.

	Algorithm	DDPG				TD3				SAC				PPO			
	Reward shifting	-100		+100		-100		+100		-100		+100		-100		+100	
	Use TD-based RC	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
No resets	Swimmer	32.03	-113.27	-3.89	-104.86	-12.03	-103.63	-3.35	-103.20	-45.95	-117.02	-32.34	-108.30	9.90	-103.27	5.42	-101.52
	HumanoidStandup	-14.40	44.69	-14.32	-29.16	-18.89	41.30	-6.92	5.70	19.61	-32.88	9.26	-24.10	10.76	-1.48	-0.03	-11.97
	Reacher	5.86	-295.90	1.75	-429.94	1.77	-112.49	1.64	-160.87	0.44	-32.00	0.37	-117.87	0.10	-3.09	-0.28	-8.67
	Pusher	1.16	-73.46	-0.63	-183.44	1.48	-86.38	0.20	-162.26	0.26	-20.45	-1.29	-25.07	-1.75	-10.79	-1.47	-19.53
	SpecialAnt	-11.15	-156.21	-12.52	-100.50	-8.91	-65.57	-9.20	-44.65	45.39	-12.97	48.23	-12.73	-2.72	-40.94	-4.95	-42.30
Predefined resets	HalfCheetah	14.53	-31.38	4.67	-59.69	-6.82	-10.51	7.35	-85.34	8.83	-45.64	3.54	-44.86	0.62	-45.83	-0.71	-62.95
	Ant	1.76	-93.29	-0.99	-118.93	-2.23	-73.80	-1.96	-97.01	-0.34	-46.73	1.26	-75.70	2.09	-34.96	6.04	-31.90
	Hopper	-0.26	-41.21	0.49	-62.35	0.16	-49.28	2.83	-53.05	-2.09	-28.59	-2.05	-17.77	-3.06	-44.08	0.26	-36.00
	Humanoid	0.60	-69.68	-0.35	-83.17	-8.97	-87.61	-10.87	-113.79	-1.70	-79.30	-0.43	-109.34	-0.60	-44.44	-1.74	-50.57
	Walker2d	-0.66	-26.45	-0.41	-63.41	-3.15	-32.44	-7.01	-53.33	-3.01	-25.64	-2.07	-40.23	-2.54	-56.70	0.24	-52.50
Agent-controlled resets	HalfCheetah	4.33	-29.17	5.82	-73.96	1.13	-34.89	-3.03	-26.31	-2.78	-46.60	-4.77	-59.21	-0.64	-78.67	13.89	-78.05
	Ant	0.59	-81.13	2.41	-127.31	2.93	-75.54	3.49	-85.00	0.09	-49.99	-1.52	-82.81	-2.20	-71.29	-0.72	-69.68
	Hopper	2.54	-23.70	-0.60	-106.48	-2.55	-34.18	-1.81	-65.56	2.45	-21.67	3.20	-11.30	-0.75	-69.86	0.85	-36.35
	Humanoid	-39.22	-180.58	-9.35	-106.27	2.30	-118.38	-2.99	-108.54	2.22	-15.29	5.33	-35.35	-0.99	-15.79	0.93	-21.05
	Walker2d	-4.87	-41.19	-0.88	-59.64	-11.85	-6.78	3.81	-46.72	-1.10	-31.97	6.78	-35.39	2.16	-72.69	-2.23	-77.86

Table 16: TD-based reward centering is not sensitive to reward shifting.

It is worth noting that we tested only the simplest choice for the f function—using the mean of action values from a fixed batch of state-action pairs. Other choices for the f function could potentially yield better results. Additionally, recent work has proposed estimating the reward rate by applying a moving average to the f function's output (Hisaki and Ono, 2024). Future research is needed to evaluate alternative choices for the f function and evaluate the moving-average approach within the reference-state-based framework.

	Algorithm	DDPG			TD3			SAC			PPO		
	RC approach	TD	RVI	MA	TD	RVI	MA	TD	RVI	MA	TD	RVI	MA
No resets	Swimmer	109.11	66.96	176.30	90.71	-59.05	-22.94	1149.26	809.21	481.01	71.14	-86.18	34.98
	HumanoidStandup	41.67	29.68	28.42	19.79	14.47	16.46	35.83	-16.14	7.40	19.39	5.30	26.30
	Reacher	-0.03	-74.41	-0.01	0.07	-78.37	0.03	-0.11	-0.14	-0.08	1.17	1.26	1.20
	Pusher	10.87	-16.80	10.70	1.24	-17.14	1.00	0.39	-2.58	-0.20	3.72	0.09	2.92
	SpecialAnt	12.67	-4.02	21.23	2.05	-9.34	-3.01	5.59	-18.81	5.15	10.55	9.59	4.38
Predefined resets	HalfCheetah	3.15	0.25	4.07	13.13	5.24	4.65	5.05	1.37	0.43	4.66	-0.14	12.69
	Ant	22.22	-14.27	13.10	18.25	-19.80	23.04	6.75	6.12	9.33	13.36	4.32	1.36
	Hopper	2.53	-20.79	1.52	14.83	-16.84	15.37	4.44	3.83	4.13	4.56	-2.31	3.23
	Humanoid	210.54	163.26	213.03	89.68	52.56	70.91	77.54	61.66	51.39	11.29	16.17	9.27
	Walker2d	16.28	0.20	7.87	10.37	-11.10	8.64	7.72	6.79	6.22	7.37	5.44	15.34
Agent-controlled resets	HalfCheetah	2.21	1.89	1.53	17.45	19.86	13.88	-2.05	-9.81	-18.64	4.06	13.00	30.94
	Ant	12.73	12.90	3.99	94.13	58.32	104.65	34.57	30.92	26.04	8.07	-8.31	-4.47
	Hopper	42.28	17.31	30.28	15.72	16.55	8.12	4.60	-3.18	5.04	5.57	2.87	7.26
	Humanoid	246.73	126.40	115.99	20.71	14.63	23.19	5.46	4.01	-3.28	2.36	1.98	0.32
	Walker2d	10.23	-7.59	-0.10	12.92	-10.86	0.79	0.89	-17.88	-5.99	4.61	0.19	7.29
	Average improvement	49.54	18.73	41.86	28.07	-2.72	17.65	89.06	57.02	37.86	11.46	-2.45	10.20

Table 17: Performance improvement when applying reward centering to the tested algorithms to solve the Mujoco testbed. Here, RVI standards for the reference-state-based approach. MA standards for the moving-average-based approach.

Task	DQN			SAC			PPO		
RC approach	TD	RVI	MA	TD	RVI	MA	TD	RVI	MA
Breakout	-7.48	-31.97	-6.79	1.67	2.10	-0.22	11.51	0.73	3.12
Pong	0.50	-33.64	2.45	51.94	50.46	-0.13	79.18	30.13	68.90
SpaceInvader	20.97	15.93	12.93	0.29	-2.54	0.73	19.72	35.18	7.96
BeamRider	7.01	6.82	3.59	35.00	13.05	0.67	75.67	4.88	72.31
Seaquest	26.79	-4.38	19.42	22.75	-10.35	0.24	5.77	-12.69	-4.65
MsPacman	9.96	8.70	4.58	1.76	-0.42	1.50	2.67	-0.61	2.41
Average improvement	9.63	-6.42	6.03	18.90	8.72	0.465	32.42	9.60	25.01

Table 18: The performance improvement when applying reward centering in the tested algorithms to solve the Atari testbeds.

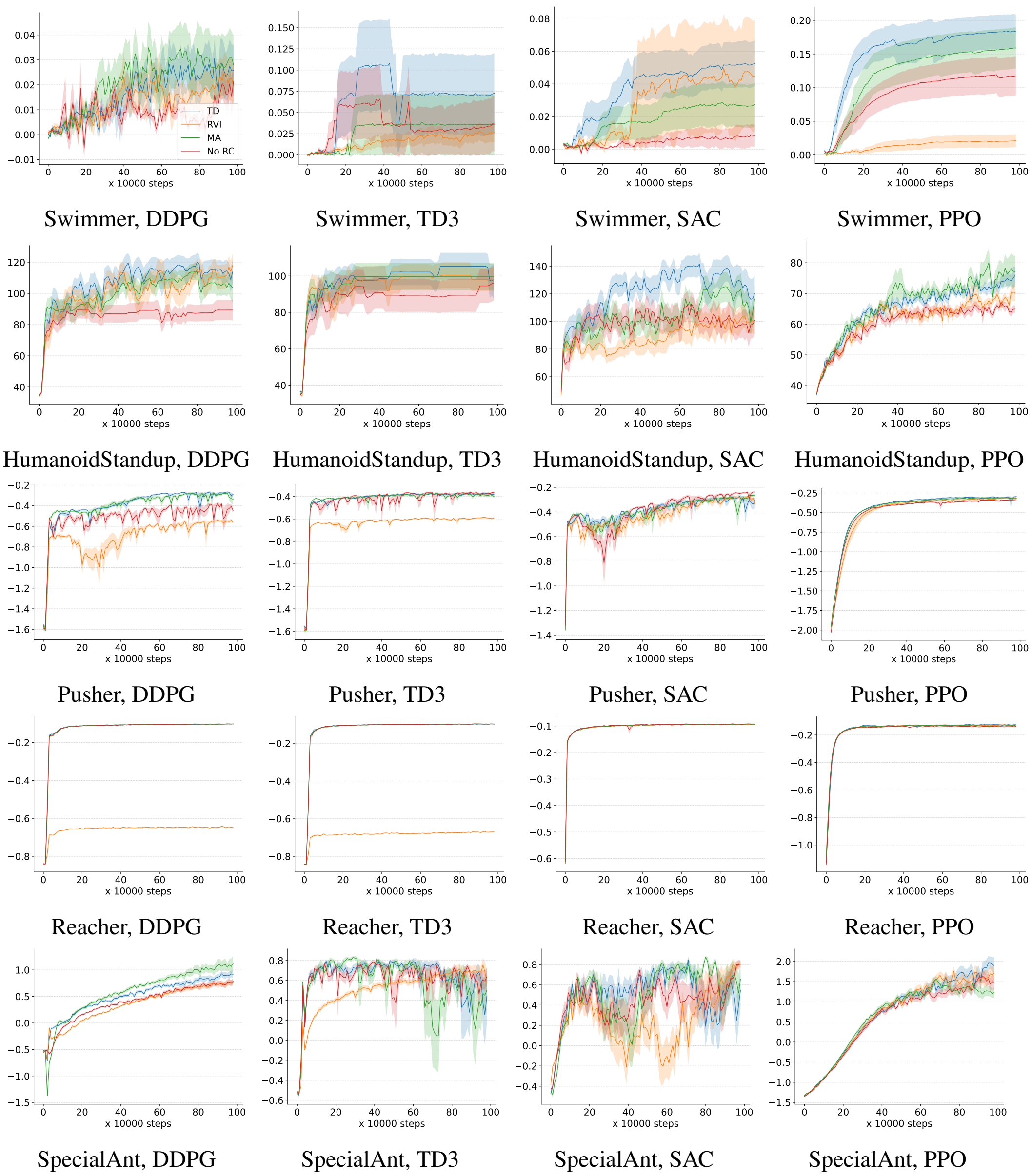

Figure 4: Learning curves on continuing testbeds without resets based on Mujoco environments. Each point shows the reward rate averaged over the past 10,000 steps.

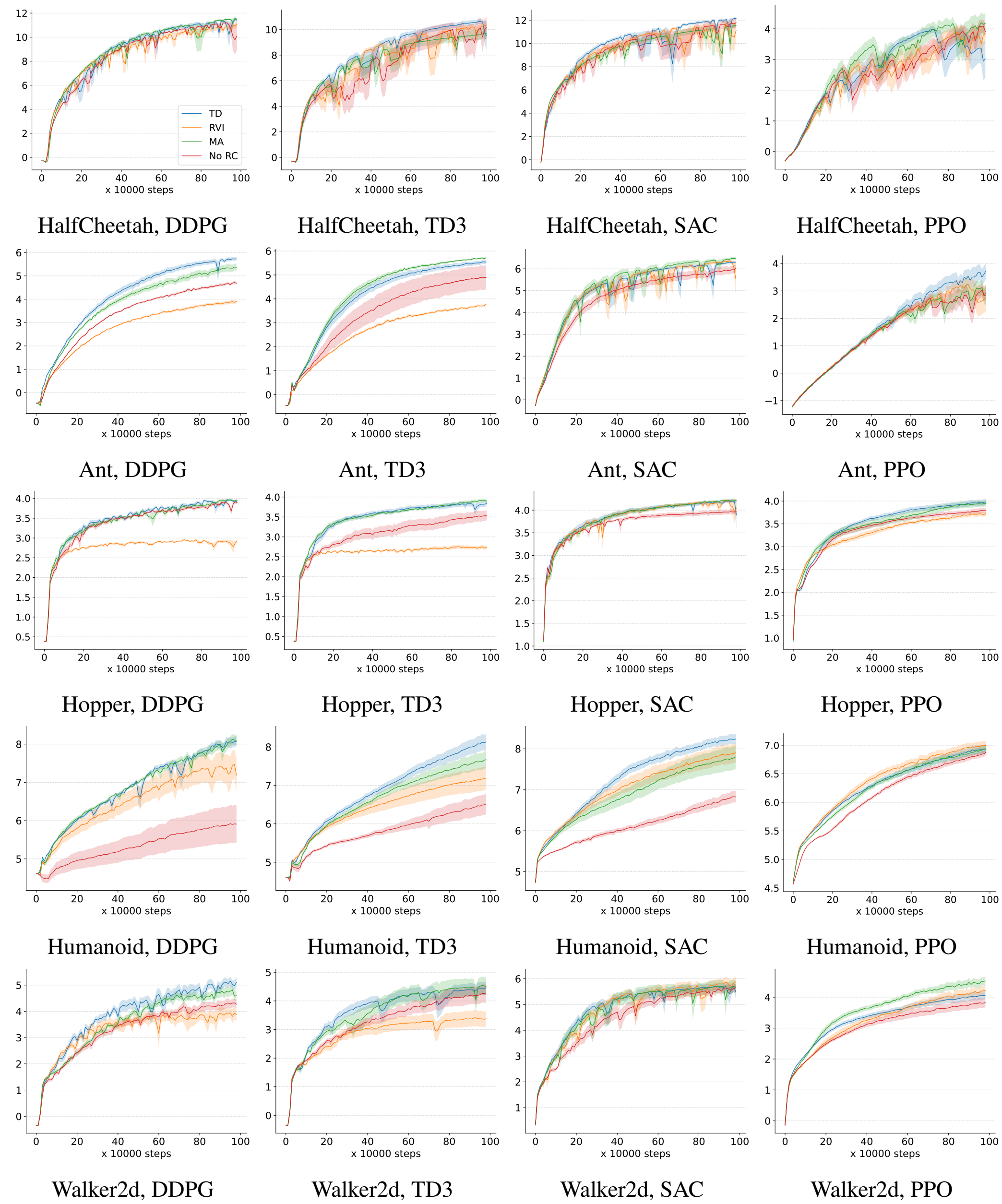

Figure 5: Learning curves on continuing testbeds with predefined resets based on Mujoco environments. Each point shows the reward rate averaged over the past 10,000 steps. The shading area standards for one standard error.

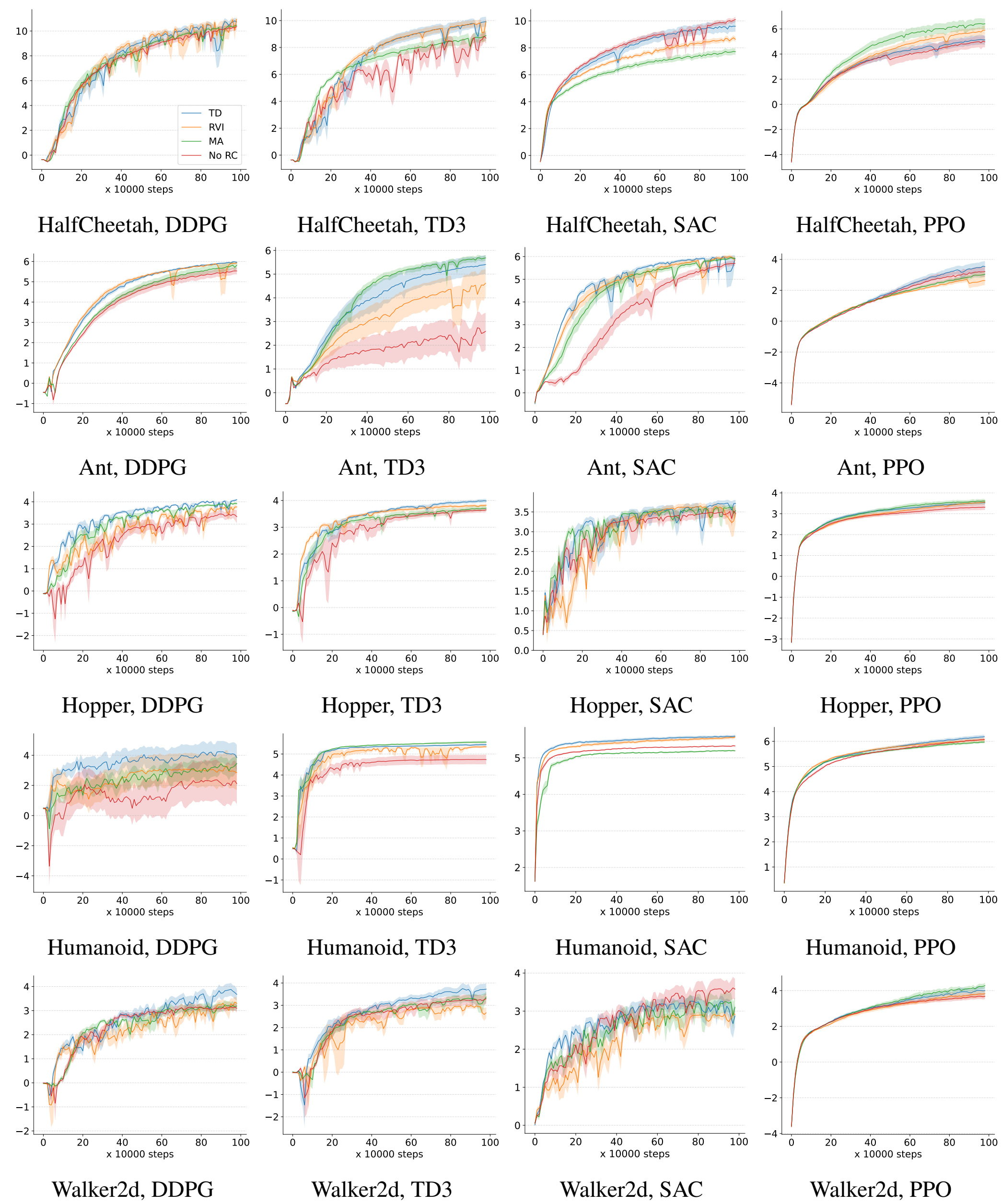

Figure 6: Learning curves on continuing testbeds with agent-controlled resets based on Mujoco environments. Each point shows the reward rate averaged over the past 10,000 steps. The shading area standards for one standard error.

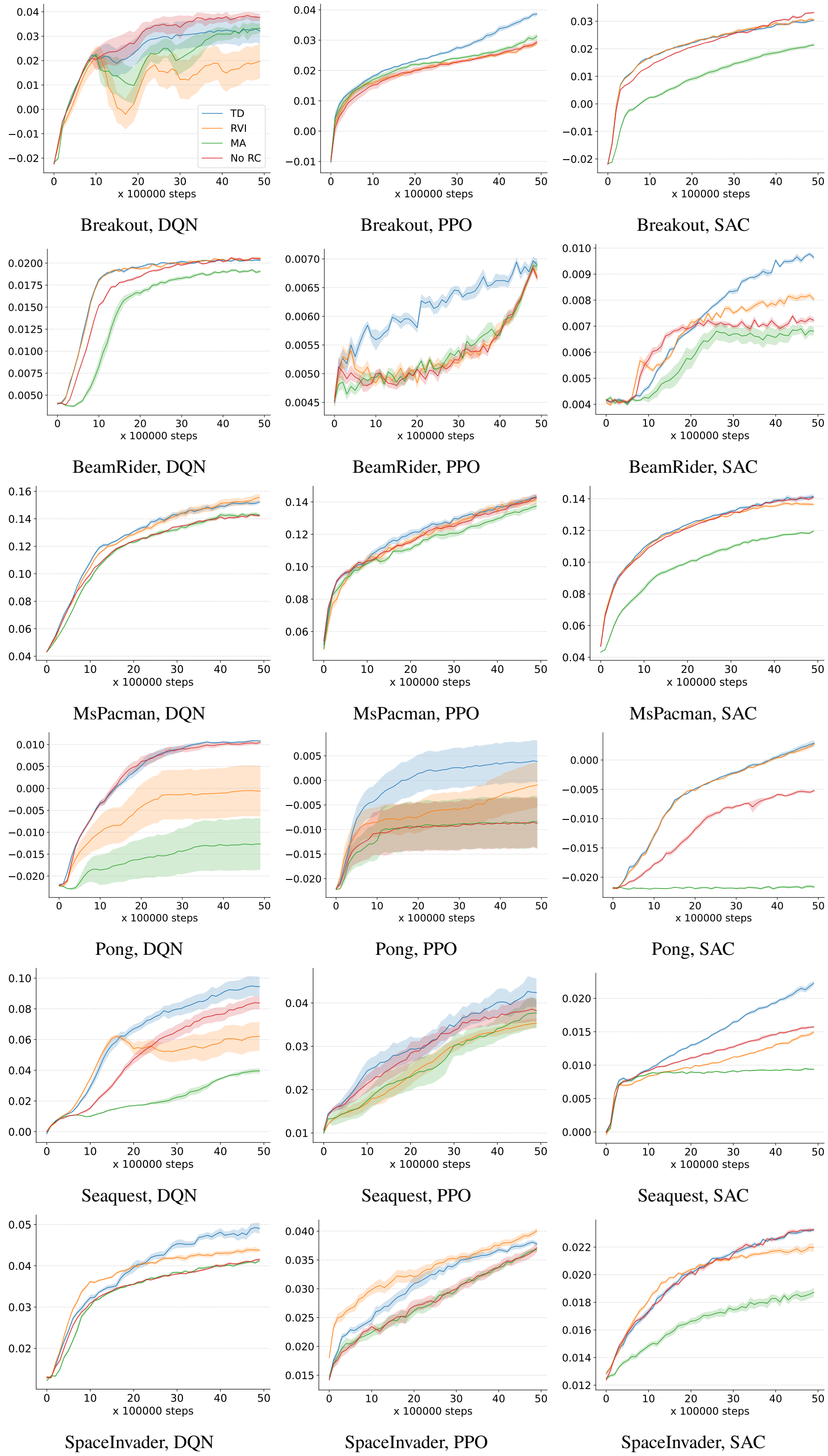

Figure 7: Learning curves on continuing testbeds with predefined resets based on Atari environments. Each point shows the reward rate averaged over the past 10,000 steps. Shading area standards for one standard error.