CAN AGENTS RUN RELAY RACE WITH STRANGERS? GENERALIZATION OF RL TO OUT-OF-DISTRIBUTION TRAJECTORIES

Li-Cheng Lan[1] **Huan Zhang**[2] **Cho-Jui Hsieh**[1]

[1]University of California, Los Angeles [2]Carnegie Mellon University

lclan@cs.ucla.edu huan@huan-zhang.com chohsieh@cs.ucla.edu

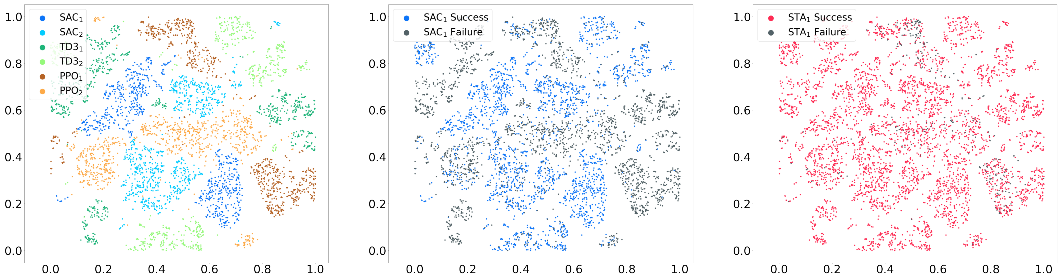
ABSTRACT

In this paper, we evaluate and improve the generalization performance for reinforcement learning (RL) agents on the set of “controllable” states, where good policies exist on these states to achieve the goal. An RL agent that generally masters a task should reach its goal starting from any controllable state of the environment instead of memorizing a small set of trajectories. To practically evaluate this type of generalization, we propose *relay evaluation*, which starts the test agent from the middle of other independently well-trained *stranger* agents' trajectories. With extensive experimental evaluation, we show the prevalence of *generalization failure* on controllable states from stranger agents. For example, in the Humanoid environment, we observed that a well-trained Proximal Policy Optimization (PPO) agent, with only 3.9% failure rate during regular testing, failed on 81.6% of the states generated by well-trained stranger PPO agents. To improve "relay generalization," we propose a novel method called Self-Trajectory Augmentation (STA), which will reset the environment to the agent's old states according to the Q function during training. After applying STA to the Soft Actor Critic's (SAC) training procedure, we reduced the failure rate of SAC under relay-evaluation by more than three times in most settings without impacting agent performance and increasing the needed number of environment interactions. Our code is available at https://github.com/lan-lc/STA.

1 INTRODUCTION

Generalization is critical for deploying reinforcement learning (RL) agents into real-world applications. A well-trained RL agent that can achieve high rewards under restricted settings may not be able to handle the enormous state space and complex environment variations in the real world. There are many different aspects regarding the generalization of RL agents.

While many existing works study RL generalization under environment variations between training and testing Kirk et al. (2021), in this paper, we study the generalization problem in a simple and *fixed* environment under an often overlooked notion of generalization — a well-generalized agent that masters a task should be able to start from any “controllable” state in this environment and still reach the goal. For example, a self-driving system may need to take over the control from humans (or other AIs trained for different goals such as speed, gas-efficient, or comfortable) in the middle of driving and must continue to drive the car safely. We can make little assumptions about what states the cars are at when the take-over happens, and the self-driving agent must learn to drive generally. Although this problem may look ostensibly easy for simple MDPs (e.g., a 2-D maze), most real RL agents are trained by trajectories generated by its policy, and it is hard to guarantee the behavior of an agent on all “controllable” states. Roughly speaking, in the setting of robotics (e.g., Mujoco environments), we can define a state as controllable if there exists at least one policy that can lead to a high reward trajectory or reach the goal from this state. Unfortunately, most ordinary evaluation procedures of RL nowadays do not take this into consideration, and the agents are often evaluated from a fixed starting point with a very small amount of perturbation (Todorov et al., 2012; Brockman et al., 2016). In fact, finding these controllable states themselves for evaluation is difficult.



(a) States from ordinary evaluation

(b) Relay-evaluation for SAC$_1$

(c) Relay-evaluation for our STA$_1$

Figure 1: t-SNE of states from trajectories of 6 Humanoid agents. (a) The states of 6 agents are almost non-overlapping even for the ones trained by the same algorithm; (b) SAC$_1$ agent in (a) performs badly on controllable states from other stranger agents, indicating that it may not learn to control the robot generally; (c) our STA agent performs uniformly well when starting from the same set of states.

The **first contribution** of this work is to propose *relay-evaluation*, a proxy to evaluate agent generalization performance on controllable states in a fixed environment. Relay-evaluation involves running an agent from the middle states of other independently trained agents' high-reward trajectories. This is similar to running a relay race with another *stranger agent*, where the stranger agent controls the robot first, and the test agent takes the reins of the robot later. It naturally finds a diverse set of controllable states for the test agents because the sampled states come from high reward trajectories of well-trained agents. The stranger agents can be trained using a variety of RL algorithms, not limited to the one used for the test agent.

Our extensive experiments on 160 agents trained in four environments using four algorithms show that many representative RL algorithms have unexpectedly high failure rates under relay-evaluation. For example, in the Humanoid environment, Proximal Policy Optimization (PPO) agents and Soft Actor Critic (SAC) agents have average failure rates of 81.6% and 38.0% under relay-evaluation even when the stranger agents have trained with the *same* algorithms (different random seeds), which is surprisingly high compared to the original failure rates (PPO: 3.9%, SAC: 0.92%). The failure of the agents under this setting shows that they may not genuinely understand the dynamics of the environments and learn general concepts like balancing the robot to avoid failing, but rather memorize a small set of actions specifically for a limited number of states it encountered. In Figure 1a, we illustrate the t-SNE for states in trajectories of 6 agents trained with 3 algorithms and observe that even trained with the same algorithm, the states generated by different agents are quite distinctive. Figure 1b shows that the SAC$_1$ agent only has a low failure rate on its own states, which is the dots colored blue in Figure 1a.

Our **second contribution** is to propose a novel training method called Self-Trajectory Augmentation (STA) that can significantly improve agents' generalization without significantly increasing training costs. We first conduct a motivative experiment to augment the initial state set of an agent during training by the states generated by a set of pretrained stranger agents, and we find that as we increase the number of stranger agents, the relay-evaluation of the agent improves significantly. However, pretraining additional models is time-consuming and may be impractical in complex environments. Therefore, we propose a novel method called Self-Trajectory Augmentation (STA), where we randomly set the agent to start from its old trajectories. Since the distribution of visited states often varies during training, reviewing an agent's old trajectories can be beneficial for generalization. After applying STA to the standard SAC training procedure, the failure rates of SAC agents are reduced by more than three times in most settings without sacrificing agent performance under ordinary evaluation, and it has minimal impact on convergence speed. In Figure 1c, our STA agent is uniformly successful on most states from 6 stranger agents.

2 Relay Evaluation

In this section, we first introduce the definition of relay-evaluation in Sec.2.1. Then we conduct extensive experiments to evaluate the relay-generalization of representative RL algorithms in Sec.2.3.

2.1 Notations

Single-player environments of RL are normally formalized as a Markov decision process (MDP). It can be defined by a tuple $M = \langle \mathcal{S}, \mathcal{A}, \mathbb{T}, R, d_0 \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathbb{T}(s_{t+1}|s_t, a_t) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is the transition function that indicates the probability of reaching state s_{t+1} after playing action a_t on state s_t; $\mathcal{R}(s_t, a_t) : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function, and $d_0(s_0)$ is the distribution of the initial state s_0. With d_0, we can define the initial state set $\mathcal{S}_0$ as $\forall s \in \mathcal{S}_0 \Leftrightarrow d_0(s) > 0$. We define an agent's policy as $\pi(s)$ which simply outputs the best action. For each $\pi(s)$, we define its the trajectories distribution as $\mathcal{T}^\pi$. Note that, a trajectory $\tau \sim \mathcal{T}^\pi$ is a list of tuple $[(s_0, a_0, r_0, s_1), (s_1, a_1, r_1, s_2), \ldots, (s_{T-1}, a_{T-1}, r_{T-1}, s_T)]$, where $s_0 \in \mathcal{S}_0$ and for each tuple $\mathbb{T}(s_{t+1}|s_t, a_t) > 0$ and $\mathcal{R}(s_t, a_t) = r_t$. We named the return of a trajectory τ as the sum of all the rewards of a trajectory $\sum_{i=0}^{T-1} r_i$, where T is the length of the trajectory and r_i is the reward of playing a_i at s_i.

2.2 Problem Definition

The goal of relay-evaluation is to evaluate test agent π_{test} performance on any "controllable" state in the state space $\mathcal{S}$. Here, we define "controllable" state as:

Definition 2.1 (Controllable states) *A state $s \in \mathcal{S}$ is controllable if there exists a trajectory $\tau = \{\ldots, (s_t = s, a_t, r_t, s_{t+1}), \ldots\}$ where s is one of its states s_t, and the trajectory has a high return $\sum_{i=0}^{T-1} r_i$ and $t <= T - L$. Here T is the length of the trajectory, L is the number of steps the agent runs after state s. It is used to ensure it is possible to play L time steps starting from s without being terminated by the environment.*

Particularly, we focus on the case of *catastrophic failures* when starting with these controllable states. In our setting of continuous control environments, we define catastrophic failure as the environment being terminated by the simulator (e.g., the agent completely falls down), but one can also define catastrophic failure as getting an extremely low return. Conceptually, controllable states are those "decent" states where catastrophic failures should not happen. We only evaluate the test agent on controllable states since there may not exist a high return policy for a random state since some states are unrecoverable.

To find controllable states, we propose to use independently well-trained agents (which we refer to as "stranger" agents) that can generate high return trajectories. Therefore, in the relay-evaluation, we evaluate a test agent π_{test} with the controllable states on the trajectories of another agent π_{gen}. We first use π_{gen} to generate M trajectories and select $\eta \times M$ of them with top returns, where $\eta \in [0, 1]$. We then sample states $s_t, t \in \{1, 2, \ldots, T - L\}$ from those high return trajectories and regard those states to be controllable. If the test agent can continue playing for L time steps without failing from a state s_t, then we say that the test agent passes the relay-evaluation on this state s_t. After testing π_{test} on all the controllable states generated by π_{gen}, we can then compute the failure rate and average return. Note that we focus on the **failure rates** in most of the tables since it's surprising that most agents encounter unexpected high failure rates under relay-evaluation, while the average return is also reported in some cases to reflect the average performance. For a test agent, we conduct relay-evaluation on multiple generating agents, including agents independently trained by the same or different algorithms, as detailed below.

2.3 Results of Relay-Evaluation on existing algorithms

Experiment setup. We conduct our experiments in four Mujoco environments in OpenAI Gym: Humanoid-v3, Walker2d-v3, Hopper-v3, and Ant-v3 with the standard setting of 1,000 steps. We do not include HalfCheetah-v3 since the simulator does not terminate even when the robot has already fallen over, so there is no standard way to determine catastrophic failures. We first select three popular algorithms, including Soft Actor-Critic (SAC) (Haarnoja et al., 2018), Twin Delayed DDPG (TD3) (Fujimoto et al., 2018), Proximal Policy Optimization (PPO) (Schulman et al., 2017), as well as two robust training methods aiming to improve agent performance under perturbed observations, SA-PPO and ATLA-PPO (Zhang et al., 2020; 2021). Note that the robustness of SA-PPO and ATLA-PPO agents are not aligned with the generalization setting in our paper, but they are included to see if there exists a connection between robustness and generalization. All the agents have a

Table 1: Failure rates (%) of relay-evaluation using states generated by stranger agents trained with 4 algorithms, reported in the 4 rows for each environment. The "Reference" column shows the failure rate of the stranger agents, serving as the baseline failure rate for these controllable states. Although SAC agents achieve the lowest failure rate, they are still quite high compared to the reference in many environments. TD3 and PPO sometimes have over 90% generalization failure rates.

Environment	Stranger Algorithm	Reference (%)	Test Agent Algorithm (Failure Rate %)			
			SAC	TD3	PPO	SA/ATLA PPO
Humanoid	SAC	0.92 ± 1.78	**38.0 ± 33.9**	83.9 ± 17.6	83.9 ± 16.5	65.1 ± 31.8
	TD3	0.62 ± 1.34	**33.6 ± 28.5**	60.5 ± 30.1	78.4 ± 20.0	67.5 ± 29.9
	PPO	3.91 ± 4.91	**48.8 ± 30.1**	77.8 ± 24.3	81.6 ± 19.1	63.2 ± 30.9
	SA-PPO	0.12 ± 0.48	83.8 ± 18.0	96.2 ± 5.66	92.9 ± 11.9	**77.0 ± 26.4**
Walker2d	SAC	0.78 ± 2.49	**26.9 ± 23.5**	35.9 ± 28.5	87.0 ± 11.4	88.9 ± 11.7
	TD3	0.31 ± 0.68	**22.7 ± 21.7**	36.9 ± 28.1	84.3 ± 14.0	87.0 ± 12.8
	PPO	0.00 ± 0.00	**14.5 ± 12.9**	25.0 ± 20.0	70.1 ± 21.0	75.1 ± 19.3
	ATLA-PPO	0.64 ± 3.20	**19.9 ± 18.1**	29.4 ± 24.0	71.1 ± 21.7	76.2 ± 20.0
Hopper	SAC	0.37 ± 0.87	**32.6 ± 36.0**	44.1 ± 21.1	62.1 ± 19.8	40.8 ± 30.4
	TD3	0.53 ± 1.36	31.9 ± 36.5	**19.8 ± 22.1**	70.2 ± 17.6	43.8 ± 28.9
	PPO	0.85 ± 2.57	**23.8 ± 33.5**	34.5 ± 18.0	56.5 ± 20.7	36.8 ± 29.6
	ATLA-PPO	0.01 ± 0.02	**30.6 ± 35.0**	31.4 ± 20.8	64.1 ± 18.4	39.5 ± 31.6
Ant	SAC	0.63 ± 0.85	**2.70 ± 1.98**	8.98 ± 9.96	4.00 ± 2.68	3.79 ± 3.09
	TD3	1.11 ± 1.43	**2.90 ± 2.66**	10.0 ± 11.6	3.98 ± 2.13	4.25 ± 3.61
	PPO	1.03 ± 1.25	**2.62 ± 1.92**	9.95 ± 8.44	3.52 ± 2.07	3.45 ± 3.23
	ATLA-PPO	1.26 ± 1.75	**2.32 ± 1.67**	7.58 ± 5.31	3.13 ± 1.96	3.76 ± 3.57

decent average return under the ordinary evaluation which samples the initial state s_0 according to distribution d_0. We independently train $N = 10$ agents for each algorithm in each environment. Every agent takes turns being the test agent and the stranger agent of other agents. For each agent π_{gen}, we first generate $M = 200$ trajectories and keep the top 100 trajectories that have a higher return. For each high-return trajectory, we sample $K = 5$ states for other agents to test.

Generalization Results of Existing Algorithms. For each agent, we conduct relay-evaluation using the trajectories generated by all the other agents as stranger agents, trained by either the same or different algorithms. To investigate the failure rate of test agents trained by Algorithm A when testing the trajectories of stranger agents trained by Algorithm B, we report the average failure rate for each A-B pair in Table 1. Each row in Table 1 shows the results of stranger agents trained by a certain algorithm B. The "Reference" column shows the failure rates of the stranger agents starting from their own controllable state, which is very low. This indicates that these states are indeed controllable. For the remaining columns, we show the failure rates of relay-evaluation, and clearly, we observe that all the agents have significantly higher failure rates than the reference failure rates. For example, in Humanoid, SAC agents have a 38% of failure rate on other SAC agents' trajectories and even higher failure rates when the stranger agent is trained by PPO or SA-PPO. In general, SAC agents have the lowest failure rate across all environments. This is probably due to the entropy term added to their goal function, which promotes exploration. However, according to our experiments, their state distribution still collapses into a small set of successful trajectories (Figure 1a) and can not handle other agents' successful trajectories properly. SA-PPO and ATLA-PPO are robustly-trained agents that perform well under perturbations on state observations. This is reflected in its smaller failure rate on Humanoid and Hopper. However, they perform badly when taking the reins from other agents, showing relay generation is quite different from adversarial robustness since we care about all controllable states instead of slightly perturbed states. Additionally, we observe that Ant-v3 is relatively easy because the robot has four legs, which is easier to keep the robot from falling. The hardest environment is Humanoid-v3 since it has the most complicated robot to balance. The failure rate of TD3 even reaches 96.2% on SA-PPO's trajectories.

Why do agents fail? To investigate why the agents perform poorly on relay evaluation, we visualize the state distribution generated by different agents. We select the best two Humanoid agents of SAC, TD3, and PPO with the highest average return from the same agents used in Table 1. We generate 200 trajectories for each agent and sample $5,000$ states from them. We then use t-SNE (Van der Maaten & Hinton, 2008) to reduce the dimension of all the states into 2D and visualize them in Fig. 1a. We color each dot according to which agent generate the states. For example, the best and the second best SAC agent' states are colored red and blue. We can see that the states of the six agents are well separated, which means that their distributions are very different, even when trained with the same algorithms. The results hold the same in all the environments. This observation suggests that the state

Table 2: Detecting agent failure by calculating $Q(s, \pi(s))$. The numbers under the "Failure States" and "Success States" columns show the results of states that the SAC test agents have failed or succeeded on. The "Reference Return" ("SAC Return") shows the average returns of the stranger agents (test agents, respectively). Q is consistently lower on failure states.

Environment	Stranger Algorithm	Failure States			Success States		
		Reference Return	SAC Return	$Q(s, \pi(s))$	Reference Return	SAC Return	$Q(s, \pi(s))$
Humanoid	SAC	2829 ± 143	319 ± 294	103 ± 136	2743 ± 96	2733 ± 104	305 ± 182
	TD3	2698 ± 114	388 ± 306	145 ± 144	2688 ± 115	2746 ± 110	261 ± 166
	PPO	2939 ± 179	302 ± 233	81 ± 109	2822 ± 171	2734 ± 110	270 ± 191
	SA-PPO	3428 ± 175	226 ± 127	50 ± 59	3288 ± 92	2758 ± 118	140 ± 161
Walker2d	SAC	3139 ± 292	386 ± 368	298 ± 135	2979 ± 213	2834 ± 257	394 ± 112
	TD3	3118 ± 350	404 ± 362	316 ± 133	2974 ± 365	2835 ± 259	411 ± 109
	PPO	2558 ± 250	312 ± 354	236 ± 109	2504 ± 247	2739 ± 236	334 ± 106
	ATLA-PPO	2673 ± 375	275 ± 301	195 ± 117	2511 ± 321	2721 ± 236	318 ± 114
Hopper	SAC	1872 ± 66	1032 ± 488	337 ± 47	1872 ± 62	1872 ± 69	355 ± 29
	TD3	1850 ± 95	801 ± 668	331 ± 55	1835 ± 96	1873 ± 72	355 ± 34
	PPO	1819 ± 78	1055 ± 506	329 ± 39	1808 ± 79	1854 ± 71	344 ± 31
	ATLA-PPO	1889 ± 96	924 ± 558	330 ± 52	1870 ± 107	1867 ± 72	352 ± 36

distribution generated by different agents is quite different, with almost no overlap. Therefore all the other agents' trajectories are out-of-distribution for a particular agent.

How do the agents fail? We directly observe how the agents control the robot in Mujoco to understand how the agents fail and the implications of passing the relay-evaluation. Fig. 2 shows the snapshots of some different SAC agents. We observe that the agents normally maintain the balance of the upper part of the robot by fixing it to a special posture. However, different agents' postures are different, and they use different ways to keep the balance. Therefore, when an agent takes over the control from another agent, it needs to correct the posture to its own. This indicates that if an agent can pass the relay-evaluation, it generally knows how to correct the robot without losing balance, even if the starting posture is unseen.

Can the agents detect that it is going to fail? Out-of-distribution (OOD) detection has long been an important question in both RL and supervised learning, and we hope to identify the states where the agents may fail using value functions. We conduct this experiment on SAC since it has the best performance in most of the settings. Since SAC itself does not include a state value function, we use $Q(s, \pi(s))$ to serve as its evaluation of a given state, where Q and π are its action-value function and its policy. The results shown in Table 2 are separated into two parts. The "Failure States" part shows the results of states that the test SAC agents failed on, and the states that the test SAC agent passed is shown in "Success States". The "Reference Return" and "SAC Ret" columns show the average return of the stranger agent and the test SAC agent on the states for extra $L = 500$ steps. The "SAC Q" shows the output $Q(s, \pi(s))$. We can see that the $Q(s, \pi(s))$ of the failure states is lower than the success states. We also notice that when the SAC agents succeed, they can perform almost as well as the stranger agents. However, the prediction $Q(s, \pi(s))$ of the success state is way lower than the prediction $Q(s, \pi(s))$ on its own states: Humanoid: 590 ± 21, Walker2d: 522 ± 73, Hopper: 363 ± 20, and Ant: 480 ± 24. This indicates that the Q function $Q(s, \pi(s))$ gives some indications about OOD and failing states. We will use this information to design our algorithms.

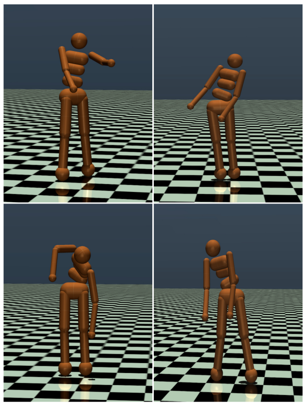


Figure 2: Different SAC agents running with different postures in Humanoid.

3 Improving the Generalization to Other Agents Trajectories

In this section, we aim to improve the relay-generalization of the existing algorithms. We first investigate a naive method in Section 3.1. Next, based on the investigation, we proposed our Self-

Trajectory Augmentation (STA) in Section 3.2. Finally, we conduct the experiments to evaluate both methods in Section 3.3.

3.1 A Naive Method: Leveraging Pretrained Agents' Trajectories

To improve relay-generalization, a naive way is to let the agent "see" at least some of the trajectories generated by other agents during training. One straightforward solution is to add the other agents' tuples (s_t, a_t, r_t, s_{t+1}) into the replay buffer as training data. However, most RL algorithms are not able to handle out-of-distribution actions except offline RL (Levine et al., 2020). Therefore, our naive method lets the agent "see" other agents' trajectories by letting the agent have chances to start playing from the controllable states of some pretrained agents during training. Before training, we first pretrain additional N_a agents. Next, we use states of those agents' top $\eta_{\text{naive}} = 0.75$ trajectories as controllable states $\mathcal{S}^{\eta_{\text{naive}}}_{\text{pretrain}}$. During training, whenever the agent starts to generate a new trajectory, besides generating a new trajectory normally, it has p_0 probability to uniformly sample a state s_{pretrain} from $\mathcal{S}^{\eta_{\text{naive}}}_{\text{pretrain}}$ as the initial state of this new trajectory. Starting from s_{pretrain}, once the agent successfully plays L_r steps, we will consider it knows how to play from s_{pretrain} and let the agent start a new trajectory. Our naive method is similar to training in an augmented environment that has a more diverse initial set state. Therefore, we do not need to modify the training algorithm.

3.2 Self-Trajectory Augmentation (STA)

Although the naive method can improve the generalization of the agents (Section 3.3), training additional agents is too time-consuming. Hence, we propose Self-Trajectory Augmentation (STA), which only uses the states of the agent's own historical trajectories to augment the initial set. The framework STA algorithm is shown in Alg. 1. For the variables, Q, π are the agent's action value and policy that need to be trained; $\mathcal{D}$ is the replay buffer to store the training data; $\mathcal{S}_{\text{STA}}$ is the set that stores the historical states during training; d_0 is the original initial state distribution. At each iteration, with probability $1 - p_0$ (line 5-7) we generate the trajectory as usual, which means sampling s_0 from d_0 and letting the agent plays for $L_{\max}$ steps (e.g., in Mujoco, $L_{\max} = 1,000$). Otherwise, with probability p_0 (line 9-10), we select a state form $\mathcal{S}_{\text{STA}}$ and let the agent plays for L_r steps. Since the failures usually happen early after taking over from another trajectory, we found a small $L_r \ll L_{\max}$ is sufficient for improving relay generalization. After generating the trajectory, we will store the trajectory into the replay buffer (line 12) and store the qualified state (line 13) into $\mathcal{S}_{\text{STA}}$ (line 14).

Algorithm 1 Our STA algorithm

```
 1: **Variable:** $Q$, $\pi$, $\mathcal{S}_{\text{STA}}$, $\mathcal{D}$
 2: **Parameter:** $p_0$, $L_r$, $d_0$
 3: **for** each iteration **do**
 4:     x$\sim$ $U(0,1)$
 5:     **if** x > $p_0$ **then**
 6:         $s_0 \sim d_0$
 7:         $\tau$ = gen_trajectory($s_0$, $L_{\max}$)
 8:     **else**
 9:         $s_0$ = select_state($\mathcal{S}_{\text{STA}}$)
10:         $\tau$ = gen_trajectory($s_0$, $L_r$)
11:     **for** s in $\tau$ **do**
12:         $\mathcal{D} = \mathcal{D} \cup \{s\}$
13:         **if** is_qualified(s) **then**
14:             $\mathcal{S}_{\text{STA}} = \mathcal{S}_{\text{STA}} \cup \{s\}$
15:     train $Q$, $\pi$ with $\mathcal{D}$
```

Definition of qualified state (Alg. 1 line 13) Ideally, a state s is qualified if it is controllable. However, during training, we cannot simply use the return of the trajectory to define if its middle states are controllable. This is mainly due to the exploration during training – the agent may play pretty well at the beginning but ends up having a low return because the algorithm tries to explore a bad action. In this case, we still want to include the beginning states into $\mathcal{S}_{\text{STA}}$. Hence, we use the sum of the next $\lambda = 50$ rewards as the scoring function score$(s_t) = \sum_{i=t}^{t+\lambda-1} r_i$ to measure whether a state is controllable. If a state's score is among the top η_{STA} ratio of the states generated in the latest epoch $\mathcal{S}_{\text{latest}}$, the state will be added into $\mathcal{S}_{\text{STA}}$. For convenience, we maintain a threshold ω so that η_{STA} ratio of the states in $\mathcal{S}_{\text{latest}}$ has a higher score than ω. Finally, we define the function is_qualified(s) in line 13 as if score$(s) > \omega$. Note that if the rewards of the environment are sparse, one might consider other scoring functions.

Select a state from $\mathcal{S}_{\text{STA}}$ (Alg. 1 line 9). We select a state from $\mathcal{S}_{\text{STA}}$ with two steps. First, we random sample N_c states that score$(s) \times \gamma \geq \omega_{\max}$ form $\mathcal{S}_{\text{STA}}$ as candidates, where $\omega_{\max}$ is the largest ω encountered during training. Although we only add states with score$(s) > \omega$ into $\mathcal{S}_{\text{STA}}$,

Table 3: The failure rates (%) of the naive baseline with different numbers N_a of pretrained agents. Training with the states from more pre-trained agents lead to better relay-evaluation performance.

Environment	Stranger Algorithm	Test Algorithm (Failure Rate %)					
		SAC	Naive$_{N_a=1}$	Naive$_{N_a=2}$	Naive$_{N_a=4}$	Naive$_{N_a=8}$	Naive$_{N_a=16}$
Humanoid	SAC	38.0 ± 33.9	55.7 ± 16.9	41.5 ± 19.7	24.0 ± 14.7	**11.4 ± 7.21**	13.2 ± 6.84
	TD3	33.6 ± 28.5	40.0 ± 13.9	22.9 ± 11.5	15.0 ± 7.96	**7.29 ± 2.79**	7.81 ± 4.76
	PPO	48.8 ± 30.1	59.3 ± 13.9	41.5 ± 12.5	30.6 ± 12.0	**24.8 ± 3.89**	25.0 ± 3.36
	SA-PPO	83.8 ± 18.0	90.8 ± 4.31	78.8 ± 5.83	71.9 ± 11.8	59.9 ± 9.60	**58.1 ± 8.32**
Walker2d	SAC	26.9 ± 23.5	28.6 ± 6.53	20.0 ± 14.6	8.95 ± 5.99	8.14 ± 5.89	**6.10 ± 2.34**
	TD3	22.7 ± 21.7	21.0 ± 6.46	17.2 ± 11.3	7.71 ± 4.31	4.71 ± 3.64	**3.43 ± 1.52**
	PPO	14.5 ± 12.9	14.0 ± 5.83	10.1 ± 10.7	3.48 ± 1.66	3.05 ± 2.14	**1.95 ± 1.35**
	ATLA-PPO	19.9 ± 18.1	19.3 ± 4.70	12.0 ± 9.00	5.46 ± 3.59	3.83 ± 2.24	**3.36 ± 1.94**
Hopper	SAC	32.6 ± 36.0	19.8 ± 15.34	10.2 ± 6.23	8.64 ± 5.50	**4.01 ± 3.76**	4.35 ± 4.73
	TD3	31.9 ± 36.5	19.1 ± 19.51	5.10 ± 4.26	11.5 ± 7.75	5.00 ± 4.51	**3.76 ± 5.74**
	PPO	23.8 ± 33.5	17.3 ± 15.77	8.57 ± 7.30	6.38 ± 4.13	**2.81 ± 2.93**	3.14 ± 4.57
	ATLA-PPO	30.6 ± 35.0	22.6 ± 22.65	5.31 ± 3.47	9.66 ± 7.71	6.46 ± 5.77	**3.67 ± 3.31**

as the agent becomes stronger, ω will also increase. Some of the states in $\mathcal{S}_{\text{STA}}$ will be considered uncontrollable since their score are too low. Hence, we use a ratio $\gamma >= 1$ to avoid low-score states ($< \omega_{\max}/\gamma$) being our candidates. For the second step, among the candidates, we want to select a state with which the current agent is unfamiliar. Based on the results in Table 2, we use $Q(s, \pi(s))$ to predict which state is harder for the current agent. That is, we select the state with the lowest $Q(s, \pi(s))$ from the candidates as the new initial state (line 9).

3.3 Experimental Results

In this section, we evaluate the agents trained by our methods under the same settings and strange agents we used in Table 1. Since SAC performs best in Table 1, we only apply our methods based on it. In the following paragraphs, we first show that the naive method can improve the relay-generalization of SAC. Next, we show that our STA can also improve the SAC agent without slowing the training process. Moreover, STA is two times better than the naive method on the hardest controllable states.

The naive method. In this experiment, we evaluate whether using pretrained models can improve the relay-generalization of an agent. For the experimental settings: we add the controllable states in the top $\eta_{\text{naive}} = 0.75$ ratio of each pretrain agent's trajectories into the starting set; we use $L_r = 100$ and $p_0 \approx 0.9$ to increase the sampling frequency of the starting set. The results of using different numbers N_a of pretrained agents are shown in Table 3. Although the pretrained agent is trained by SAC, we observe that the failure rates on all kinds of algorithms decrease as more pretrained agents are included in $\mathcal{S}^{\eta_{\text{naive}}}_{\text{pretrain}}$. This indicates that the method can successfully improve relay generalization. However, in Humanoid, the performance has converged when $N_a = 8$. We think it is because Humanoid is so complex that adding more agents still cannot cover all the playing styles. Hence, the failure rates do not keep dropping after $N_a \geq 8$. We also notice that when only using 1 pretrained agent ($N_a = 1$), the failure rate surprisingly increased in the Humanoid environment; this might be because the agent will overfit the only pretrain agent's trajectories. Additional results like using different $L_r = \{50, 100, 200, 500\}$ are shown in the Appendix.

Self-Trajectory Augmentation (STA). In this experiment, we compare the base SAC, our naive baseline with 16 pretrained agents, and our STA method with tuned parameters (see Appendix). Note that the environment step number of training each agent is all three million. Therefore, the training time of STA will be almost 17 times faster than the naive method.

The comparison of both the average return and failure rate are shown in Table 4. We first show the performance under ordinary evaluation ($s_0 \sim d_0$) of each environment ("ordinary" rows). The results show that both the naive method and STA have almost equal or even higher ordinary performances, although agents are trained under augmented MDP, which is different from the original MDP. Moreover, the failure rates of STA in the ordinary setting are also lower than SAC in all the environments, which means that the STA agents are more robust. Next, we show the relay-evaluation results on different stranger algorithms. According to the rest of the rows (except the "ordinary" row) in Table 4, we notice that STA has three times lower failure rates compared to SAC in most of the settings. In addition, since we only sample $N_c = 5$ candidates, the training time is almost the same as SAC. Hence, we claim that we achieve a better relay-generalization than SAC without losing the ordinary performance and the converging speed. We also compare the results between STA

Table 4: The returns and the failure rates of SAC, our naive baseline, and STA under relay-evaluation. Since all the agents are trained with three million environment interactions, training an STA agent is 15.9 times faster than a Naive$_{N_a=16}$ agent. The "ordinary" rows show that STA will not decrease the regular performance. The rows except "ordinary" show that STA can achieve significantly lower failure rates under relay-evaluation compared to SAC and achieve performance similar to Naive$_{N_a=16}$.

Environment	Stranger Algorithm	SAC		Naive$_{N_a=16}$ (ours)		STA (ours)	
		Return	Failure Rate	Return	Failure Rate	Return	Failure Rate
Humanoid	ordinary	5650 ± 238	0.62 ± 0.79 %	5695 ± 159	2.65 ± 2.52 %	**5899 ± 346**	**0.45 ± 0.61** %
	SAC	2004 ± 1193	38.0 ± 33.9 %	2591 ± 841	13.2 ± 33.8 %	2815 ± 622	**5.77 ± 23.3** %
	TD3	2242 ± 1071	33.6 ± 28.5 %	2731 ± 621	7.81 ± 26.8 %	2862 ± 504	**3.76 ± 19.0** %
	PPO	1675 ± 1266	48.8 ± 30.1 %	2266 ± 1120	25.0 ± 43.3 %	2692 ± 793	**9.95 ± 29.9** %
	SA-PPO	671 ± 996	83.8 ± 18.0 %	1382 ± 1343	58.1 ± 49.3 %	2312 ± 1198	**25.2 ± 43.4** %
Walker2d	ordinary	5717 ± 445	0.16 ± 0.52 %	**5986 ± 77**	**0.00 ± 0.00** %	5736 ± 422	**0.00 ± 0.00** %
	SAC	2286 ± 1138	26.9 ± 23.5 %	2931 ± 658	**5.38 ± 22.5** %	2784 ± 725	7.33 ± 26.0 %
	TD3	2326 ± 1078	22.7 ± 21.7 %	2965 ± 537	**3.71 ± 18.9** %	2822 ± 632	5.48 ± 22.7 %
	PPO	2464 ± 884	14.5 ± 12.9 %	2931 ± 336	**1.19 ± 10.8** %	2798 ± 473	2.57 ± 15.8 %
	ATLA-PPO	2314 ± 1007	19.9 ± 18.1 %	2888 ± 449	**2.33 ± 15.0** %	2782 ± 488	2.83 ± 16.5 %
Hopper	ordinary	3662 ± 137	5.75 ± 8.85 %	3609 ± 60	**0.34 ± 0.89** %	**3686 ± 58**	0.42 ± 0.77 %
	SAC	1777 ± 390	32.6 ± 36.0 %	1846 ± 148	**2.59 ± 15.8** %	1818 ± 323	9.80 ± 29.7 %
	TD3	1654 ± 594	31.9 ± 36.5 %	1838 ± 265	**3.05 ± 17.1** %	1790 ± 428	13.8 ± 34.5 %
	PPO	1790 ± 337	23.8 ± 33.5 %	1832 ± 140	**2.62 ± 15.9** %	1829 ± 245	8.52 ± 27.9 %
	ATLA-PPO	1711 ± 491	30.6 ± 35.0 %	1824 ± 257	**4.69 ± 21.1** %	1841 ± 276	10.2 ± 30.3 %

Table 5: Failure rates (%) of STA with different N_c, where N_c is the number of starting candidate states we consider before starting a trajectory. State with the lowest $Q(s, \pi(s))$ is chosen among the N_c candidates to start the agent. A larger N_c produces better performance with negligible cost.

Environment	Stranger Algorithm	Test Algorithm (Failure Rate %)				
		SAC	Naive$_{N_a=16}$	STA$_{N_c=1}$	STA$_{N_c=2}$	STA$_{N_c=5}$
Humanoid	SAC	38.0 ± 33.9	13.2 ± 6.84	19.7 ± 18.1	8.52 ± 5.74	**5.77 ± 3.06**
	TD3	33.6 ± 28.5	7.81 ± 4.76	9.29 ± 7.94	**3.48 ± 2.89**	3.76 ± 4.49
	PPO	48.8 ± 30.1	25.0 ± 3.36	26.1 ± 10.6	15.7 ± 5.25	**9.95 ± 5.38**
	SA-PPO	83.8 ± 18.0	58.1 ± 8.32	64.2 ± 8.57	48.5 ± 7.44	**25.2 ± 6.47**
Walker2d	SAC	26.9 ± 23.5	5.38 ± 3.58	20.1 ± 12.2	16.6 ± 9.47	**9.62 ± 3.72**
	TD3	22.7 ± 21.7	3.71 ± 2.48	16.3 ± 10.3	13.1 ± 6.16	**10.0 ± 5.81**
	PPO	14.5 ± 12.9	1.19 ± 0.86	12.1 ± 9.35	8.95 ± 6.09	**5.90 ± 2.64**
	ATLA-PPO	19.9 ± 18.1	2.33 ± 0.99	15.8 ± 9.78	10.4 ± 7.28	**7.67 ± 3.90**
Hopper	SAC	32.6 ± 36.0	2.59 ± 2.41	**13.61 ± 10.2**	19.8 ± 16.0	14.1 ± 8.12
	TD3	31.9 ± 36.5	3.05 ± 5.03	22.1 ± 21.0	20.9 ± 17.3	**17.6 ± 12.3**
	PPO	23.8 ± 33.5	2.62 ± 2.95	**12.6 ± 14.3**	18.1 ± 18.1	12.8 ± 12.5
	ATLA-PPO	30.6 ± 35.0	4.69 ± 4.01	24.2 ± 20.5	23.5 ± 21.8	**21.2 ± 16.0**

and the naive method. We find out that our method has a better performance in harder environments. For example, STA has only 25.2% failure rate on the Humanoid SA-PPO. However, in the easiest environment, Hopper, the naive method with $N_a = 16$ pretrained agents has a twice lower rate than STA. We suggest it is because Hopper has the simplest robot that only has one lag. Hence, we suggest that most of the hoping styles have a similar version in the 16 pretrained agents. Therefore, the naive method can easily play on other agents' trajectories.

Besides showing the best performance of STA, we also conduct some ablation studies. First, we directly observe how STA takes over the control of other agents. Same as SAC agents, we notice that STA agents also run with one kind of posture. Even when the STA starts from other agents' posture, it will change it into its own and keep running. We consider this a good thing. For example, if a test agent is faster than the stranger agent, we will want the test agent to run in its own way, which is faster. We also conduct an experiment on different numbers of candidates N_c to show the importance of starting the trajectories from the states that has a lower $Q(s, \pi(s))$. The results are shown in Table 5. We first look at the column where $N_c = 1$. It shows the results of simply randomly selecting a state from the initial set as we did in the naive method. Compared to SAC, $N_c = 1$ is only slightly better. However, as we use a larger N_c, the result becomes comparable to the naive method, especially on Humanoid. We also notice that different N_c have little effect on Hopper. This is because there is only a small $Q(s, \pi(s))$ difference between the familiar states and the unfamiliar state according to Table 2. Hence, we should consider other methods in Hopper.

4 Related Work

Many methods have been designed to evaluate the generalization of an RL agent (Kirk et al., 2021). They can be categorized by which part of the RL environment are different between training and testing, including (1) the environment, (2) the observation emission function, (3) the transition function, (4) the reward function, and (5) the initial state set. For **(1)**, researchers studied training and testing in totally different environments (Cobbe et al. (2019), and Dennis et al. (2020)). For **(2)**, researchers change the surface of the objects in the environment (Tobin et al., 2017; Zhang et al., 2018b; Zhao et al., 2019; James et al., 2019). For **(3)**, researchers change the parameter of the transition function like the gravity strength or the weight of the ball, or add an external force to the agent. (Packer et al., 2018; Pinto et al., 2017) For the **(4)**, researchers (Mishra et al., 2018; Dosovitskiy et al., 2017; Chevalier-Boisvert et al., 2019; Lynch & Sermanet, 2021) normally want to see if the agent can adapt quickly from one task to another with the general knowledge learned in the first task. Finally, the last category **(5)**, which only selects different initial states in the same environment (Portelas et al., 2019; Peng et al., 2018; Kaplanis et al., 2018) is where our relay-generalization belongs. In one of the most related previous work, Zhang et al. (2018a) only train on limited states in the initial set d_0 and test on the other states in the initial set d_0. Their results show that if a PPO agent only trains on less than 5 states, it will be hard to avoid failing on other states in the initial set d_0. In addition, our relay-generalization belongs to Out-of-Distribution (OOD) generalization, where training and testing states are sampled from different distributions (See Figure 1a and Figure 3). To the best of our knowledge, the notion of controllable states and relay evaluation has not been discussed in the literature.

Beyond the generalization of RL, many other RL domains are related to our works. First, in hierarchical reinforcement learning, Pateria et al. (2021); Brochu et al. (2010); Nachum et al. (2018) train multiple agents with different small missions and let them complete a task together by taking turns. However, in such a scenario, the agents are trained together. Therefore, they are familiar with previous agents' trajectories. Second, both our naive method and the offline RL (Levine et al., 2020; Kumar et al., 2020; Ostrovski et al., 2021) leverage the trajectories of other agents. Third, population-based training (Jaderberg et al., 2017; Derek & Isola, 2021) can be a potential way to generate controllable states and increase relay-generalization. Finally, the adversarial robustness of agents has been studied in the context of state, observation, and reward perturbations (Huang et al., 2017; Ilahi et al., 2021; Zhang et al., 2020; 2021; Ding et al., 2022; Lan et al., 2022). However, these studies focus on the performance of agents under imperceptible adversarial perturbations. On the other hand, the states of other stranger agents can be very different from the agent's trajectory distribution. In addition, our work shows that robust agents (SA/ATLA PPO) are still bad at generalization, which is similar to some of the previous results (Korkmaz, 2021).

Revisiting observed states strategy (or local access protocol (Yin et al., 2021)) is used in our STA algorithm. This strategy has been used in many RL algorithms. Ecoffet et al. (2019) surpasses the state-of-the-art on on hard-exploration games by revisiting promising states in history. Tavakoli et al. (2020) shows that revisiting states on high return trajectories can improve PPO on a sparse-reward task. After our publication, Yin et al. (2023) proposed revisiting the historical states that the agent is uncertain. Different from these works, our STA algorithm uses the Q function to select states that the agent used to be good at and now is not confident with.

5 Conclusion

We propose relay-evaluation, a proxy to evaluate RL agents' generalization performance on controllable states in a fixed environment. Relay-evaluation involves running an agent from the middle states of other independently-trained agents' high-reward trajectories. Extensive studies under the MuJoCo environment demonstrate that many representative RL algorithms have unexpectedly high failure rates under relay-evaluation, indicating a significant limitation of existing RL methods. To overcome this challenge, we propose a novel method called Self-Trajectory Augmentation (STA), which improves the generalization of existing RL agents without impacting ordinary performance. This paper opens up a new research direction in RL, and there are several limitations to be addressed in the future. For instance, how to find unexplored controllable states more effectively and how to "certify" that an agent can work (or at least won't fail) on all controllable states are interesting future challenges. More importantly, how to develop better algorithms to improve relay generalization is still an open problem.

ACKNOWLEDGEMENT

This work is supported in part by NSF under IIS-2008173, IIS-2048280 and an Okawa research grant.

REFERENCES

Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *ArXiv*, abs/1012.2599, 2010.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *ArXiv*, abs/1606.01540, 2016.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. In *ICLR*, 2019.

Karl Cobbe, Oleg Klimov, Christopher Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. *ArXiv*, abs/1812.02341, 2019.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre M. Bayen, Stuart J. Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *ArXiv*, abs/2012.02096, 2020.

Kenneth Derek and Phillip Isola. Adaptable agent populations via a generative model of policies. *Advances in Neural Information Processing Systems*, 34:3902–3913, 2021.

Qin Ding, Cho-Jui Hsieh, and James Sharpnack. Robust stochastic linear contextual bandits under adversarial attacks. In *International Conference on Artificial Intelligence and Statistics*, pp. 7111–7123. PMLR, 2022.

Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. Carla: An open urban driving simulator. *ArXiv*, abs/1711.03938, 2017.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *ArXiv*, abs/1901.10995, 2019.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep rl: A case study on ppo and trpo. In *International conference on learning representations*, 2019.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *ArXiv*, abs/1802.09477, 2018.

Tuomas Haarnoja, Aurick Zhou, P. Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

Inaam Ilahi, Muhammad Usama, Junaid Qadir, Muhammad Umar Janjua, Ala Al-Fuqaha, Dinh Thai Hoang, and Dusit Niyato. Challenges and countermeasures for adversarial attacks on deep reinforcement learning. *IEEE Transactions on Artificial Intelligence*, 3(2):90–109, 2021.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12619–12629, 2019.

Christos Kaplanis, Murray Shanahan, and Claudia Clopath. Continual reinforcement learning with complex synapses. *ArXiv*, abs/1802.07239, 2018.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktaschel. A survey of generalisation in deep reinforcement learning. *ArXiv*, abs/2111.09794, 2021.

Ezgi Korkmaz. Adversarial training blocks generalization in neural policies. 2021.

Aviral Kumar, Aurick Zhou, G. Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *ArXiv*, abs/2006.04779, 2020.

Li-Cheng Lan, Huan Zhang, Ti-Rong Wu, Meng-Yu Tsai, I Wu, Cho-Jui Hsieh, et al. Are alphazero-like agents robust to adversarial perturbations? *arXiv preprint arXiv:2211.03769*, 2022.

Sergey Levine, Aviral Kumar, G. Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *ArXiv*, abs/2005.01643, 2020.

Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data. *Robotics: Science and Systems XVII*, 2021.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and P. Abbeel. A simple neural attentive meta-learner. In *ICLR*, 2018.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *NeurIPS*, 2018.

Georg Ostrovski, Pablo Samuel Castro, and Will Dabney. The difficulty of passive learning in deep reinforcement learning. *Advances in Neural Information Processing Systems*, 34:23283–23295, 2021.

Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Xiaodong Song. Assessing generalization in deep reinforcement learning. *ArXiv*, abs/1810.12282, 2018.

Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8, 2018.

Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Kumar Gupta. Robust adversarial reinforcement learning. In *ICML*, 2017.

Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. In *CoRL*, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

Arash Tavakoli, Vitaly Levdik, Riashat Islam, Christopher M. Smith, and Petar Kormushev. Exploring restart distributions. *arXiv: Learning*, 2020.

Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30, 2017.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Dong Yin, Botao Hao, Yasin Abbasi-Yadkori, Nevena Lazi’c, and Csaba Szepesvari. Efficient local planning with linear function approximation. *ArXiv*, abs/2108.05533, 2021.

Dong Yin, Sridhar Thiagarajan, Nevena Lazic, Nived Rajaraman, Botao Hao, and Csaba Szepesvari. Sample efficient deep reinforcement learning via local planning. *ArXiv*, abs/2301.12579, 2023.

Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *ArXiv*, abs/1806.07937, 2018a.

Amy Zhang, Yuxin Wu, and Joelle Pineau. Natural environment benchmarks for reinforcement learning. *ArXiv*, abs/1811.06032, 2018b.

Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Mingyan D. Liu, Duane S. Boning, and Cho-Jui Hsieh. Robust deep reinforcement learning against adversarial perturbations on state observations. *arXiv: Learning*, 2020.

Huan Zhang, Hongge Chen, Duane S. Boning, and Cho-Jui Hsieh. Robust reinforcement learning on state observations with learned optimal adversary. *ArXiv*, abs/2101.08452, 2021.

Chenyang Zhao, Olivier Sigaud, Freek Stulp, and Timothy M. Hospedales. Investigating generalisation in continuous deep reinforcement learning. *ArXiv*, abs/1902.07015, 2019.

A Details for Experimental Setup

For SAC and TD3, we follow the implementation of OpenAI Spinning Up (Achiam, 2018), which uses networks of size (256, 256) with relu units and 3 million environment interactions. For PPO, we follow the implementation of Engstrom et al., 2019 since its PPO performance is higher than OpenAI Spinning Up. The total number of environmental interactions is one million. For SA-PPO and ATLA-PPO, we follow the implementation in (Zhang et al., 2021). SA-PPO and ATLA-PPO are representative algorithms in terms of observation robustness. That is, the agent can function normally even if a small adversarial perturbation is added to the observation. We include this kind of robust agent to show that our problem is different from the normal adversarial robustness problem in RL. For the humanoid environment, the SA-PPO algorithm we used is the SGLD algorithm. For other environments, we use ATLA-PPO.

For the **naive method**, both the pretrained agents and the agent follow the implementation of SAC of OpenAI Spinning Up. Hence, training a naive method with N_a agents requires $N_a + 1$ times of training time. The default parameter settings are: $\eta_{\text{navie}} = 0.75$ since we want to include harder states; $p_0 \approx 0.9$, $L_r = 100$ since we want to sample more initial states form $\mathcal{S}_{\text{pretrain}}^{\eta_{\text{naive}}}$.

For the **STA method**, we also follow the implementation of SAC of OpenAI Spinning Up. Since the candidate number N_c we use is less than 10, the training time is almost the same as training a SAC agent. The default parameter settings are: $\eta_{\text{STA}} = 0.75$ since we want to include harder states as we did in the naive method; $p_0 \approx 0.9$, $L_r = 100$ since we want to sample more initial states form $\mathcal{S}_{STA}$; $\lambda = 50 < L_r$ since we want to include the states in the trajectories that starts from states in $\mathcal{S}_{STA}$. For the STA parameters used in Table 4, we set the number of candidates $N_c = 5$; we set the qualifying ratio $\gamma = 1.0$ in Humanoid and $\gamma = 1.6$ in Walker2d and Hopper.

B Experiments of our naive method with different playing lengths

In this section, we show the results of our naive method with different limitations of the extra steps of the trajectories starting from starting set.

The results are shown in Table 6. According to the table, the failure rate will increase when L_r is too small or too large. Since $L_r = 100$ has the lowest failure rate in almost all settings, we will use $L_r = 100$ as the default in other experiments.

Table 6: The results of the naive method with different time step limitation L_r.

Environment	Stranger Algorithm	Test Algorithm (Failure Rate %)				
		SAC	Naive$_{L_r=50}$	Naive$_{L_r=100}$	Naive$_{L_r=200}$	Naive$_{L_r=500}$
Humanoid	SAC	38.0 ± 33.9	30.4 ± 34.9	**13.2 ± 6.84**	14.3 ± 6.00	14.76 ± 5.08
	TD3	33.6 ± 28.5	25.7 ± 37.3	**7.81 ± 4.76**	10.6 ± 3.74	11.76 ± 6.61
	PPO	48.8 ± 30.1	40.6 ± 30.0	**25.0 ± 3.36**	25.0 ± 5.04	28.36 ± 7.31
	SA-PPO	83.8 ± 18.0	69.7 ± 15.7	**58.1 ± 8.32**	61.8 ± 12.2	65.43 ± 10.8
Walker2d	SAC	26.9 ± 23.5	6.10 ± 2.34	5.38 ± 3.58	**4.48 ± 3.15**	6.52 ± 2.87
	TD3	22.7 ± 21.7	3.43 ± 1.52	3.71 ± 2.48	**3.14 ± 1.80**	3.48 ± 2.20
	PPO	14.5 ± 12.9	1.95 ± 1.35	**1.19 ± 0.86**	1.81 ± 1.06	2.81 ± 1.97
	ATLA	19.9 ± 18.1	3.36 ± 1.94	**2.33 ± 0.99**	2.88 ± 1.24	3.78 ± 2.24
Hopper	SAC	32.6 ± 36.0	4.35 ± 4.73	**2.59 ± 2.41**	7.96 ± 5.68	7.82 ± 6.54
	TD3	31.9 ± 36.5	3.76 ± 5.74	**3.05 ± 5.03**	5.19 ± 4.37	6.52 ± 9.11
	PPO	23.8 ± 33.5	3.14 ± 4.57	**2.62 ± 2.95**	5.86 ± 5.15	3.71 ± 6.79
	ATLA	30.6 ± 35.0	**3.67 ± 3.31**	4.69 ± 4.01	6.05 ± 3.88	5.85 ± 7.50

C Experiments of STA with Different Candidate Criteria

In this section, we evaluate the performance of using different γ. In STA, we use ω_{max}/γ to serve as the criteria to be the threshold of being one of the N_c candidates. Only states with score ω_{max}/γ in $\mathcal{S}_{\text{STA}}$ can be the candidate of the initial state s_0. If γ is larger, then states with lower scores can also be the candidates.

The results different γ with $N_c = 5$ are shown in Table 7. According to the results, the best γ of each environment is different. For example, in Humanoid, it is important to only samples the states that have higher scores as candidates (small γ).

Table 7: The results of STA with different γ. When γ is larger, the states with lower scores can also be the candidates of being the initial state of a trajectory during training STA.

Environment	Stranger Algorithm	Test Algorithm (Failure Rate %)					
		SAC	Naive	$\gamma = 1.0$	$\gamma = 1.1$	$\gamma = 1.3$	$\gamma = 1.6$
Humanoid	SAC	38.0 ± 33.9	13.2 ± 6.84	5.77 ± 3.06	**4.44 ± 5.34**	8.36 ± 5.62	9.10 ± 5.55
	TD3	33.6 ± 28.5	7.81 ± 4.76	3.76 ± 4.49	**2.24 ± 2.67**	6.05 ± 4.30	3.62 ± 3.56
	PPO	48.8 ± 30.1	25.0 ± 3.36	**9.95 ± 5.38**	12.0 ± 6.33	15.8 ± 5.20	12.4 ± 5.61
	SA-PPO	83.8 ± 18.0	58.1 ± 8.32	**25.2 ± 6.47**	33.2 ± 15.2	48.5 ± 11.4	43.6 ± 6.97
Walker2d	SAC	26.9 ± 23.5	5.38 ± 3.58	9.62 ± 3.72	11.5 ± 6.92	**6.52 ± 3.92**	7.33 ± 4.62
	TD3	22.7 ± 21.7	3.71 ± 2.48	10.0 ± 5.81	11.5 ± 7.70	6.38 ± 5.76	**5.48 ± 3.35**
	PPO	14.5 ± 12.9	1.19 ± 0.86	5.90 ± 2.64	7.00 ± 4.10	5.24 ± 4.69	**2.57 ± 3.25**
	ATLA	19.9 ± 18.1	2.33 ± 0.99	7.67 ± 3.90	8.17 ± 5.16	7.49 ± 5.27	**2.83 ± 2.04**
Hopper	SAC	32.6 ± 36.0	2.59 ± 2.41	14.1 ± 8.12	23.6 ± 15.1	23.2 ± 10.6	**9.80 ± 4.51**
	TD3	31.9 ± 36.5	3.05 ± 5.03	17.6 ± 12.3	18.3 ± 19.2	28.5 ± 19.5	**13.8 ± 8.20**
	PPO	23.8 ± 33.5	2.62 ± 2.95	12.8 ± 12.5	20.2 ± 16.3	21.2 ± 12.6	**8.52 ± 6.04**
	ATLA	30.6 ± 35.0	4.69 ± 4.01	21.2 ± 16.0	21.0 ± 17.2	28.9 ± 18.7	**10.2 ± 9.48**

D Infinity replay buffer

In this section, we show that our STA method is better than using an unlimited-size of replay buffer. The original replay buffer size is one million. The maximum size of an unlimited-size of replay buffer is three million since we only have three million environment interactions. The results are shown in Table 8. According to the results, SAC with an infinite replay buffer still has poor performance on relay evaluation compared to our method. Moreover, the average return under ordinary evaluation is lower than the normal SAC, while our method is greater or equal to the normal SAC.

Table 8: SAC with infinity replay buffer during training.

Environment	Stranger Algorithm	Test Algorithm (Failure Rate %)			
		SAC	Naive$_{N_a=16}$	STA	SAC$_{\text{replay\_buffer}=\infty}$
Humanoid	SAC	38.0 ± 33.9	13.2 ± 6.84	5.77 ± 23.3	34.07 ± 10.12
	TD3	33.6 ± 28.5	7.81 ± 4.76	3.76 ± 19.0	24.00 ± 15.67
	PPO	48.8 ± 30.1	25.0 ± 3.36	9.95 ± 29.9	43.39 ± 12.31
	SA-PPO	83.8 ± 18.0	58.1 ± 8.32	25.2 ± 43.4	80.10 ± 8.43
Walker2d	SAC	26.9 ± 23.5	5.38 ± 3.58	7.33 ± 26.0	41.33 ± 26.87
	TD3	22.7 ± 21.7	3.71 ± 2.48	5.48 ± 22.7	38.19 ± 27.21
	PPO	14.5 ± 12.9	1.19 ± 0.86	2.57 ± 15.8	24.57 ± 23.94
	ATLA-PPO	19.9 ± 18.1	2.33 ± 0.99	2.83 ± 16.5	28.47 ± 25.46

E Using Average Reward as Scoring Function

In this section, we show the STA result of using the average score $\text{score}(s_t) = \sum_{i=t}^{T-1} r_i/(T-t)$ instead of using the reward sum of next λ steps $\text{score}(s_t) = \sum_{i=t}^{t+\lambda-1} r_i$. In STA, the scoring function is a quick way to estimate if a state is controllable. We only add the states that have high enough scores into the $\mathcal{S}_{\text{STA}}$. According to the results, using the average reward can also reduce the failure rate compared to the original SAC. However, its failure rate is still higher than using our score function (column STA), which uses the sum of the reward of the next λ step.

Table 9: The experiment of STA that use average rewards $\text{score}(s_t) = \sum_{i=t}^{T-1} r_i/(T-t)$ as the score function to estimate if a state in $\mathcal{S}_{\text{STA}}$ is controllable.

Environment	Stranger Algorithm	Test Algorithm (Failure Rate %)			
		SAC	Naive$_{N_a=16}$	STA	STA$_{\text{average reward}}$
Humanoid	SAC	38.0 ± 33.9	13.2 ± 6.84	5.77 ± 23.3	8.36 ± 3.52
	TD3	33.6 ± 28.5	7.81 ± 4.76	3.76 ± 19.0	4.86 ± 2.30
	PPO	48.8 ± 30.1	25.0 ± 3.36	9.95 ± 29.9	11.85 ± 4.72
	SA-PPO	83.8 ± 18.0	58.1 ± 8.32	25.2 ± 43.4	44.29 ± 11.14

F The failure definition of Mujoco

In Mujoco, by default, most of the environments will terminate the simulation when the agent is unhealthy. Normally, unhealthy means the robot has fallen over. By doing so, the agent can start a new trajectory without wasting more time on the "unhealthy" trajectory. Each environment has its own way of detecting unhealthy.

The Humanoid environment requires the z-position (height) of the robot need to be in a predefined range.

The Ant and Walker2d environment define the robot as unhealthy if any of the following happens:

- Any of the state space values is no longer finite.
- The height of the walker is not in a predefined range.
- The absolute value of the angle is not in a predefined range.

The Hopper environment defines the robot as unhealthy if any of the following happens:

- An element of observation[1:] is not in a predefined range.
- The height of the walker is not in a predefined range.
- The absolute value of the angle is not in a predefined range.

Please see more details at https://www.gymlibrary.dev/environments/mujoco/.

G Define Failure with Returns

In this section, we define the failure of relay-evaluation according to the reward sum of the next $L = 500$ steps instead of using the failure defined by Mujoco. The first column shows the failure rate using the failure definition of the Mujoco. For the remaining columns, we allow the agent to keep playing even if the current state is "failed," according to Mujoco. In the end, we define a trajectory as a failure trajectory according to its return. For example, for the second column, we define a trajectory as failed when its return is lower than 500. Note that the average return of SAC should be 2828. According to the results, no matter using which kind of definition of failure, SAC still has an unacceptable failure rate, especially on the SA-PPO. This shows that the termination criteria in the MuJoCo environments are not too strict and most of the terminated trajectories have no chance to get a high score.

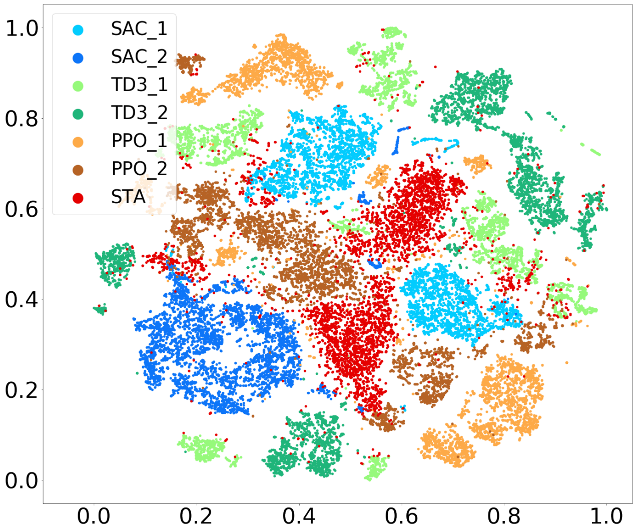
Table 10: Conducting relay-evaluation with different definitions of failure. The column $\text{SAC}_{\text{Mujoco}}$ uses the failure defined by the environment. The columns $\text{SAC}_{<\text{thr}}$ define a trajectory is failed if the return value is smaller than the threshold thr. Note that the expected return value is greater than 2800.

Environment	Stranger Algorithm	Test Algorithm (Failure Rate %)			
		$\text{SAC}_{\text{Mujoco}}$	$\text{SAC}_{<500}$	$\text{SAC}_{<1000}$	$\text{SAC}_{<1500}$
Humanoid	SAC	38.0 ± 33.9	30.56 ± 11.11	32.28 ± 11.12	32.54 ± 10.95
	TD3	33.6 ± 28.5	21.55 ± 5.77	23.93 ± 5.67	24.09 ± 5.76
	PPO	48.8 ± 30.1	42.42 ± 10.02	44.93 ± 10.41	45.33 ± 10.48
	SA-PPO	83.8 ± 18.0	81.83 ± 9.29	82.82 ± 9.03	82.98 ± 9.04

H State Distribution of an STA Agent's Old Trajectories

In this section, we plot the historical states of an STA agent with the states we showed in Fig 1a. As the figure shows, although the distribution of historical states of the STA agent is more diverse than the distribution of other agents' states, there are still many areas that it does not cover. Hence, this suggests that the STA agents have a better performance on relay evaluation due to being more general.

Figure 3: State distribution of an STA agent's $\mathcal{S}_{\text{STA}}$. $\mathcal{S}_{\text{STA}}$ stores the likely controllable states that the agent generated during training. We use t-SNE to visualize those states along with the states generated by the other six agents used in Fig.1a.



I Evaluating SAC with Different L in Relay-Evaluation

In this section, we conduct relay-evaluations with different L. We select SAC as our test agent. Our result is shown in Table 11. The results show that most of the failures happened in the first 100 steps after the test agent took over the control. The results also show that for some of the cases, even if the agent does not fail for 100 steps, it may still fail in the next 400 steps.

Table 11: Evaluating SAC with different L in relay-evaluation, where L is the extra steps that the test agent needs to complete without failing.

Environment	Stranger Algorithm	Test Algorithm (Failure Rate %)			
		SAC$_{L=500}$	SAC$_{L=200}$	SAC$_{L=100}$	SAC$_{L=50}$
Humanoid	SAC	38.0 ± 33.9	32.32 ± 11.08	30.69 ± 10.92	21.12 ± 7.25
	TD3	33.6 ± 28.5	23.97 ± 5.71	21.71 ± 6.22	11.19 ± 6.09
	PPO	48.8 ± 30.1	45.06 ± 10.59	42.50 ± 10.26	28.84 ± 7.81
	SA-PPO	83.8 ± 18.0	82.90 ± 9.03	82.02 ± 9.05	73.77 ± 9.60

J Generating Controllable States with Different Hyperparameters

In this section, our goal is to generate more controllable states to test our agents by training with different hyperparameters. We first train 40 SAC agents with random hyperparameters and use the top 10 agents to serve as stranger agents.

The hyperparameters that we will randomly choices are: the learning rate of the policy π, the learning rate of the Q function, the batch size, the network structure, the training frequency, and the α in the SAC goal function.

The results are shown in Table 12. According to Table 12, we observe that the states generated by random hyperparameters are easier for other agents to pass the relay-evaluation, especially when evaluating SAC and TD3 agents. A potential reason is that some of these hyperparameters lead to worse policies and generate slower trajectories, and it's easier for the target agent to take over from those slower states.

Table 12: We generate controllable states by training SAC with random hyperparameters, including learning rates, network sizes, and batch size.

Environment	Stranger Algorithm	Test Agent Algorithm (Failure Rate %)			
		SAC	TD3	PPO	SA-PPO/ATLA
Humanoid	SAC	**38.0 ± 33.9**	83.9 ± 17.6	83.9 ± 16.5	65.1 ± 31.8
	TD3	**33.6 ± 28.5**	60.5 ± 30.1	78.4 ± 20.0	67.5 ± 29.9
	PPO	**48.8 ± 30.1**	77.8 ± 24.3	81.6 ± 19.1	63.2 ± 30.9
	SA-PPO	83.8 ± 18.0	96.2 ± 5.66	92.9 ± 11.9	**77.0 ± 26.4**
	SAC-Random-Hyperparameters	**16.6 ± 37.2**	55.8 ± 49.6	79.5 ± 40.3	65.0 ± 47.6

K The ordinary return of the test agents in Table 1

This section shows the average returns of the agents we used in Table 1.

Table 13: The avg return of the test agents we used in Table 1.

Environment	SAC	TD3	PPO	SA/ATLA PPO
Humanoid	5645.73 ± 233.54	5011.30 ± 1515.59	5173.36 ± 236.10	6614.09 ± 340.69
Walker2d	5715.64 ± 446.39	5588.48 ± 701.70	4184.67 ± 897.59	3636.84 ± 1140.55
Hopper	3667.81 ± 143.62	3520.63 ± 177.80	5384.12 ± 144.40	4587.02 ± 271.10
Ant	6155.81 ± 176.08	6007.45 ± 770.93	3148.41 ± 404.15	2721.81 ± 900.34