Adversarial Style Transfer for Robust Policy Optimization in Deep Reinforcement Learning

Anonymous authors
Paper under double-blind review

Abstract

This paper proposes an algorithm that aims to improve generalization for reinforcement learning agents by removing overfitting to confounding features. Our approach consists of a max-min game theoretic objective. A generator transfers the style of observation during reinforcement learning. An additional goal of the generator is to perturb the observation, which maximizes the agent's probability of taking a different action. In contrast, a policy network updates its parameters to minimize the effect of such perturbations, thus staying robust while maximizing the expected future reward. Based on this setup, we propose a practical deep reinforcement learning algorithm, Adversarial Robust Policy Optimization (ARPO), to find a robust policy that generalizes to unseen environments. We evaluate our approach on Procgen and Distracting Control Suite for generalization and sample efficiency. Empirically, ARPO shows improved performance compared to a few baseline algorithms, including data augmentation.

1 Introduction

The reinforcement learning (RL) environments often provide observation, which is a high-dimensional projection of the true state, complicating policy learning as the deep neural network model might mistakenly correlate reward with irrelevant information. Thus deep neural networks might overfit irrelevant features in the training data due to their high flexibility and the long-stretched time duration of the RL training process, which leads to poor generalization (Hardt et al., 2016; Zhang et al., 2018a; Cobbe et al., 2019; 2020; Zhang et al., 2018b; Machado et al., 2018; Gamrian & Goldberg, 2019). These irrelevant features (e.g., background color) usually do not impact the reward; thus, an optimal agent should avoid focusing on them during policy learning. Even worse, this might lead to incorrect state representations, which prevent deep RL agents from performing well even in slightly different environments. Thus, to learn a correct state representation, the agent needs to avoid those features. FIX

In recent times, generalization in RL has been explored extensively. In zero-shot generalization framework (Zhang et al., 2018b; Song et al., 2020; Wang et al., 2019; Packer et al., 2018) the agent is trained on a finite training set of MDP levels and then evaluated on the full distribution of the MDP levels. The distribution of the MDP levels can be generated in various ways, including changing dynamics, exploit determinism, and adding irrelevant confounders to high-dimensional observation. An agent trained in these setups can overfit any of these factors, resulting in poor generalization performance. Many approaches have been proposed to address this challenges including various data augmentation approaches such as random cropping, adding jitter in image-based observation (Cobbe et al., 2019; Laskin et al., 2020a; Raileanu et al., 2020; Kostrikov et al., 2020; Laskin et al., 2020b), random noise injection (Igl et al., 2019), network randomization (Osband et al., 2018; Burda et al., 2018; Lee et al., 2020), and regularization (Cobbe et al., 2019; Kostrikov et al., 2020; Igl et al., 2019; Wang et al., 2020) have shown to improve generalization. The common theme of these approaches is to increase diversity in the training data so as the learned policy would better generalize. However, this perturbation is primarily done in isolation of the RL reward function, which might change an essential aspect of the observation, resulting in sub-optimal policy learning. Moreover, the random perturbation in various manipulations of observations such as cropping, blocking, or combining two random images from different environment levels might result in unrealistic observations that the agent will less FIX

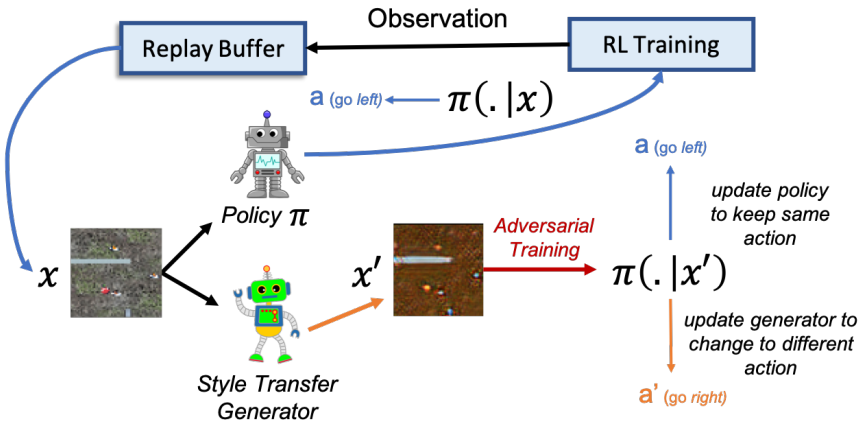


Figure 1: Overview of our approach. The method consists of a minimax game theoretic objective. It first applies a clustering approach to generate n group of clusters based on the different visual features in the observation. A generator is then used to style-translate observation from one cluster to another while maximizing the change in action probabilities of the policy. In contrast, the policy network updates its policy parameters to minimize the translation effect while optimizing for the RL objective that maximizes the expected future reward.

likely observe during testing. Thus these techniques might work poorly in the setup where agents depend on realistic observation for policy learning. It is also desirable to train the agent with realistic observations, which helps it understand the environments' semantics. Otherwise, the agent might learn unexpected and unsafe behaviors while entirely focusing on maximizing rewards even by exploiting flaws in environments such as imperfect reward design.

This paper focuses on a particular form of generalization where the agent gets to observe a high-dimensional projection of the true state. In contrast, the latent state dynamics remain the same across all the MDP levels. This assumption has been shown to affect generalization (Song et al., 2020) in well studied benchmarks (Nichol et al., 2018). It has been observed that the agent overfits the scoreboard or timer and sometimes achieves the best training performance even without looking at the other part of the observation. Consequently, this policy completely fails to generalize in test environments. The high-capacity model even can memorize the whole training environment, thus severely affecting the generalization (Zhang et al., 2018b). FIX FIX

This paper proposes a style transfer-based observation translation method that considers the observation's content and reward signal. A generator performs the style transfer while the policy tries to be robust toward such style changes. In particular, we propose a unified learning approach where the style transfer is integrated into the reinforcement learning training. Thus the generator produces more effective translation targeted toward RL objective, making the RL policy more robust. FIX

In our setup, the trajectory data from the agent's replay buffer is first clustered into different categories, and then observation is translated from one cluster's style to another cluster's style. Here the style is determined by the attribute commonality of observation features in a cluster. The agent should be robust toward changes of such features. Moreover, the translated trajectories correspond to those that possibly appear in testing environments, assisting the agent in adapting to unseen scenarios.

Figure 1 shows an overview of our method. Our approach consists of a max-min game theoretic objective where a generator network modifies the observation by style transferring it to maximize the agent's probability of taking a different action. In contrast, the policy network (agent) updates its parameters to minimize the effect of perturbation while maximizing the expected future reward. Based on this objective, we propose a practical deep reinforcement learning algorithm, Adversarial Robust Policy Optimization (ARPO), that aims to find a robust policy by mitigating the observational overfitting and eventually generalizes to test environments.

We empirically show the effectiveness of our ARPO agent in generalization and sample efficiency on challenging Procgen (Cobbe et al., 2020) and Distracting Control Suite (Stone et al., 2021) benchmark with image-based observation. The Procgen leverages procedural generation to create diverse environments to test the RL agents' generalization capacity. The distracting control adds various visual distractions on top of the Deepmind Control suite (Tassa et al., 2020) to facilitate the evaluation of generalization performance.

Empirically, we observed that our agent ARPO performs better in generalization and sample efficiency than the baseline PPO (Schulman et al., 2017) and a data augmentation-based RAD (Laskin et al., 2020a) algorithm on the many Procgen environments. Furthermore, APRO performs better compared to PPO and SAC (Haarnoja et al., 2018) on the two Distracting Control Suite environments in various setups.

In summary, our contributions are listed as follows:

- We propose Adversarial Robust Policy Optimization (ARPO), a deep reinforcement learning algorithm to find a robust policy that generalizes to test environments in a zero-shot generalization framework.
- We evaluate our approach on challenging Procgen (Cobbe et al., 2020) and Distracting Control Suite (Stone et al., 2021) benchmark in generalization ans sample efficiency settings.
- Empirically, we observed that our agent ARPO performs better generalization and sample efficiency performance compared to standard PPO (Schulman et al., 2017), SAC (Haarnoja et al., 2018), and a data augmentation-based RAD (Laskin et al., 2020a) approach in various settings.

2 Preliminaries and Problem Settings

Markov Decision Process (MDP) An MDP is denoted by $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r)$ where at every timestep t, from an state $s_t \in \mathcal{S}$, the agent takes an action a_t from a set of actions $\mathcal{A}$. The agent then receives a reward r_t from the environment and move to a new state $s_{t+1} \in \mathcal{S}$ based on the transition probability $P(s_{t+1}|s_t, a_t)$. FIX

Reinforcement Learning. Reinforcement learning aims to learn a policy $\pi \in \Pi$ that maps state to actions. The policy's objective is to maximize cumulative reward in an MDP, where Π is the set of all possible policies. Thus, the policy which achieves the highest possible cumulative reward is the optimal policy $\pi^* \in \Pi$.

Generalization in Reinforcement Learning. In the zero-shot generalization framework (Song et al., 2020), we assume the existence of a distribution of levels $\mathcal{D}$ over an MDP and a fixed optimal policy π^* which can achieve maximal return over levels generated from the distribution $\mathcal{D}$. The levels may differ in observational variety, such as different background colors. In this setup, the agent has access to a fixed set of MDP levels during training the policy. The trained agent is then tested on the unseen levels to measure the generalization performance of the agent. This scenario is often called Contextual MDP (i.e., CMDP). A detailed analysis can be found in the survey paper Kirk et al. (2021). FIX

Style Transfer. The task of image-to-image translation is to change a particular aspect of a given image to another, such as red background to green background. Generative adversarial network (GAN)-based method has achieved tremendous success in this task (Kim et al., 2017; Isola et al., 2017; Zhu et al., 2017; Choi et al., 2018). Given training data from two domains, these models learn to style-translate images from one domain to another. The domain is defined as a set of images sharing the same attribute value, such as similar background color and texture. The shared attributes in a domain are considered as the "style" of that domain. Many translation methods require paired images from two domains, which is not feasible in the reinforcement learning setup. The agent collects the data during policy learning, and annotating them in pairs is not feasible. Thus, we leverage unpaired image to image translation, which does not require a one-to-one mapping of annotated images from two domains.

In particular, we build upon the work of StarGAN (Choi et al., 2018) which is capable of learning mappings among multiple domains efficiently. The model takes in training data of multiple domains and learns the mappings between each pair of available domains using only a single generator. This method automatically captures special characteristics of one image domain and figures out how these characteristics could be

translated into the other image collection, making it appealing in the reinforcement learning setup. However, the RL agent generates experience trajectory data which is not separated into domains. Thus, we further apply a clustering approach which first clusters the trajectory observation into domains, and then we train the generator, which style-translates among those domains.

3 Adversarial Robust Policy Optimization (ARPO)

Our approach consists of a max-min game theoretic objective where a generator network modifies the observation by changing the style of it to maximize the agent's probability of taking a different action. In contrast, FIX the policy network (agent) updates its parameters to minimize the effect of perturbation while maximizing the expected future reward. Based on this objective, we propose a practical deep reinforcement learning algorithm, Adversarial Robust Policy Optimization (ARPO), to find a robust policy that generalizes to test environments. We now discuss the details of the policy network and generator.

3.1 Policy Network

The objective of the policy network is two-fold: maximize the cumulative RL return during training and be robust to the perturbation on the observation imposed by the generator. Ideally, the action probability of the translated image should not change from the original image, as the translated observation is assumed to represent the same semantic as the original observation.

The reinforcement learning loss $\mathcal{L}_\pi$ can be optimized with various reinforcement learning mechanisms, such as policy gradient (optimization). However, we do not make any assumption the type of RL algorithms to use for optimizing $\mathcal{L}_\pi$; thus, any algorithms can be incorporated with our method. Here we discuss the policy optimization formulation, which is based on the proximal policy optimization (PPO) (Schulman et al., 2017) objective:

$$\mathcal{L}_\pi = -\mathbb{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t] \tag{1}$$

$$A_t = -V(s_t) + r_t + \gamma r_{r+1} + ... + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T). \tag{2}$$

where, π_θ is the current policy and $\pi_{\theta_{old}}$ is the old policy; A_t is estimated from the sampled trajectory by policy $\pi_{\theta_{old}}$ leveraging a value function estimator denoted as V. Here, both the policy π and value network V are represented as neural networks.

An additional objective of the policy is to participate in the adversarial game with the style transfer generator. Thus the final policy loss $\mathcal{L}_\theta$ is defined as follows:

$$\mathcal{L}_\theta = \mathcal{L}_\pi + \beta_1 KL[\pi_\theta(.|x_t), \pi_\theta(.|x'_t)] \tag{3}$$

The policy parameter θ is updated to minimize equation 3. For the observation x_t at timestep t, the KL-component measure the distortion in the action distribution of the policy π_θ due to the perturbation (x'_t). The hyperparameter β_1 determines the participation in an adversarial objective with the generator. By minimizing the KL-part, the policy becomes robust to the change in perturbation proposed by the generator.

3.2 Generator Network

It takes the observation x_t as input and outputs the style-translated observation x'_t. The objective of this network is to change the agent's (policy network's) probability of taking a different action for the corresponding input observation. However, the content or semantic of the observations needs to be intact, and the only style of the images will differ.

The first step of this process is to curate a dataset with different style features such as background color and texture of objects present in the observation. We first use experience trajectory observation data from the environment and separate them into different classes based on visual features (Figure 2). The task of the generator is then to translate the images from one class images to another class images. In this case, the

"style" is defined as the commonality among images in a single class. We now discuss details about how we separate the trajectory observation to get different clusters. FIX

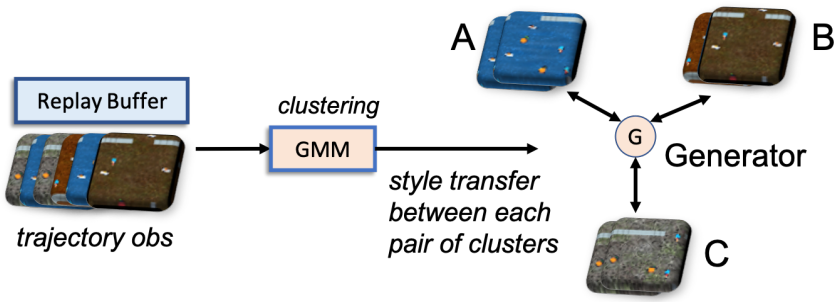


Figure 2: Overview of style transfer generator module. The experience trajectory for the task environment are separated into different classes based on visual features using Gaussian Mixture Model (GMM) clustering algorithm. The task of the generator is then to translate the image from one class image to another classes images. In this case, the "style" is defined as the commonality among images in a single class. Given a new observation, it first infers into its (source) cluster using GMM, and then the generator translated it to (target) another cluster style. The target cluster is taken randomly from the rest of the cluster.

Clustering Trajectory Observation. The trajectory data is first clustered into n clusters. Then, we use the Gaussian Mixture Model (GMM) for clustering and ResNet (He et al., 2016) for feature extraction and dimension reduction. Firstly, a pre-trained (on ImageNet) ResNet model reduces an RGB image into a 1000 dimensional vector. This step extracts useful information from the raw image and reduces the dimension, allowing faster cluster training. Note that this process focuses entirely on the visual aspect of the observation. After passing through the ResNet, the observation dataset is clustered using GMM.

Generator Training. Generator G tries to fool discriminator D in an adversarial setting by generating a realistic image represented by the true image distribution. We extend the generator's ability, and an additional objective of the generator is to fool the RL agent (policy network) by transforming the style of the original observation. For example, changing the background color to a different one does not change the semantic of the observation, but the image style would be different. Ideally, the action mapping of the policy needs to be the same on both the original and translated observation. However, a policy that focuses on the irrelevant background color will suffer from this translation, and thus its probability distribution deviates because of the style translation. This is defined as the KL-divergence $KL(\pi_\theta(.|x_t), \pi_\theta(.|x'_t))$, where x_t is the given observation and x'_t is the translated observation by the generator.

The generator tries to generate a style that eventually increases the KL quantity. On the other hand, the policy should learn to be robust toward such changes. The policy training described above addresses this and tries to minimize the KL-divergence due to the style transfer, which eventually allows robust policy learning and ultimately prevents the agent from overfitting irrelevant features from the high-dimensional observation.

We build on the generator on previous works (Choi et al., 2018; Zhu et al., 2017) which is a unified framework for a multi-domain image to image translation. The discriminator loss is

$$\mathcal{L}_D = -\mathcal{L}_{adv} + \lambda_{cls}\mathcal{L}^r_{cls}, \tag{4}$$

which consist of the adversarial loss $\mathcal{L}_{adv}$ and domain classification loss $\mathcal{L}^r_{cls}$. The discriminator detects a fake image generated by the generator G from the real image in the given class data.

On the other hand, the generator loss is

$$\mathcal{L}_G = \mathcal{L}_{adv} + \lambda_{cls}\mathcal{L}^f_{cls} + \lambda_{rec}\mathcal{L}_{rec} - \beta_2 KL(\pi_\theta(.|x_t), \pi_\theta(.|x'_t)) \tag{5}$$

which consists of image translation components and policy component which is the KL-part. From the translation side, the $\mathcal{L}_{adv}$ is adversarial loss with discriminator $\mathcal{L}^f_{cls}$ is the loss of detecting fake image. Finally, a cycle consistency loss $\mathcal{L}_{rec}$ (Choi et al., 2018; Zhu et al., 2017) is used which makes sure the FIX

translated input can be translated back to the original input, thus only changing the domain related part and not the semantic. Some more details of these losses can be found in the supplementary materials.

Our addition to the generator objective is the KL-part in equation 5 which tries to challenge the RL agent and fool its policy by providing a more challenging translation. However, the eventual goal of this adversarial game is to help the agent to be robust to any irrelevant features in the observations. The hyperparameter β_2 control the participation in the adversarial objective of the generator network.

Now we discuss a practical RL algorithm based on the above objectives.

3.3 ARPO Algorithm

The training of the policy network and the generator network happen in an alternating way. First, the policy is trained on a batch of replay buffer trajectory observation, and then the generator parameters are updated based on the objectives discussed above. The more overall training progresses, the better the generator at translating observation, and the policy also becomes better and becomes robust to irrelevant style changes. For stable training, the input images data for the generator is kept fixed initially, and we apply the GMM cluster only at the beginning to generate different class images (distribution). The pseudocode of the algorithm is given in Algorithm 1.

Algorithm 1 Adversarial Robust Policy Optimization (ARPO)

Initialize parameter vectors for policy network and generator network
for each iteration **do**
  for each environment step **do**
    $a_t \sim \pi_\theta(a_t|x_t)$
    $x_{t+1} \sim P(x_{t+1}|x_t, a_t)$
    $r_t \sim R(x_t, a_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x_t, a_t, r_t, x_{t+1})\}$
  end for
  for each observation x_t in $\mathcal{D}$ **do**
    $x'_t \leftarrow Generator(x_t)$ *// generate translated observation*
    Compute $\mathcal{L}_\pi$ from data $\mathcal{D}$ using Equation 1, and 2 *// update policy for RL objective*
    Compute $\mathcal{L}_\theta$ using Equation 3 *// update policy with adversarial KL part*
    Update generator by computing Equation 6, and 5
  end for
end for

Discussion on convergence. The adversarial min-max KL component $KL(\pi_\theta(.|x_t), \pi_\theta(.|x'_t))$ become zero when the RL policy is optimal (π^*) and the generators only perturbs (translates) the irrelevant part of all observations. In that case, the optimal policy only focuses on the relevant part of the observations, that is, the true state; thus, any changes in irrelevant part due to style transfer will be ignored. At that point the KL component become zero as the $\pi^*_\theta(.|x_t) = \pi^*_\theta(.|x'_t)$. Note that in practice, the algorithm is usually trained for limited timesteps, and thus the adversarial training might not converge to optimal. However, empirically we observe that our algorithm ARPO achieves improved performance in many challenging Procgen and Distracting control suite tasks.

4 Experiments

4.1 Setup

Our experiments cover both discrete (Procgen) and continuous (Distracting control) action spaces with image-based observation.

Procgen Environment. We evaluate our proposed agent ARPO on the challenging RL generalization benchmark Procgen (Cobbe et al., 2020). We use the standard evaluation protocol from Cobbe et al. (2020)

where the agent is allowed to train on only 200 levels of each environment, with the difficulty level set as *easy*. Then the agents are tested over more than 100K levels, full distribution. Thus, to better generalize to the test environment, the agent must master the skill and avoid overfitting to spurious features in the training environment. This setup is in-distribution generalization as a fixed number of train levels are randomly taken from an i.i.d level distribution. The observations are raw pixel images, and the environment varies drastically between levels; some snippets are given in Figure 3.

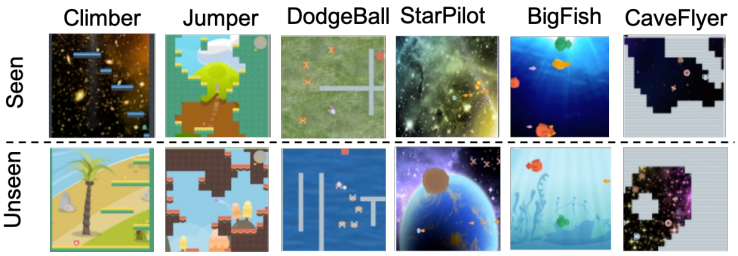


Figure 3: Some snippets of different Procgen environments (Cobbe et al., 2020). The training (seen) levels vary drastically from the testing (unseen) environment. The agent must master the skill without overfitting irrelevant non-generalizable aspects of the environment to perform better in unseen levels.

Distracting Control Suite Environment. The distracting control adds various visual distractions on top of the Deepmind Control suite (Tassa et al., 2020) to facilitate the evaluation of generalization performance. We experimented on two distracting settings, *background (dynamic)* and *color* for *easy* difficulty mode. In the distracting background, videos played in the background, and for the color distraction, the body of the robot color changes in different agent interactions. The train and test environments differ by different distractions, such as separate videos for train and test. As the agent has to use image-based observation, the distractions make the task challenging. This setup is an out-of-distribution generalization as the test time videos differ from train videos and are unseen during training. Some snippets of the environments can be found in Figure 4.

Baselines. We compare our algorithm with Proximal Policy Optimization (**PPO**), an on-policy algorithm motivated by how we can quickly optimize our policy while staying close to the current policy. PPO performs effectively in many RL benchmarks, which include the Procgen environments (Cobbe et al., 2020). FIX Furthermore, we evaluated our method with a data augmentation technique, **RAD** (Laskin et al., 2020a) on Procgen environments. RAD is a plug-and-play module that can be used with existing RL algorithms and shows effective empirical performance in complex RL benchmarks, including some Procgen environments. In particular, we evaluated *Cutout Color* augmentation technique which has shown better results in many Procgen environments compared to other augmentation techniques evaluated in Laskin et al. (2020a). This augmentation is applied on top of standard PPO. Note that our ARPO algorithm can be easily plugged in with these data augmentation-based methods. However, in this paper, we evaluated how our ARPO agent performs even when no augmentation is used. Furthermore, in the Distracting control suite, in addition to PPO, we compare ARPO with Soft Actor-Critic (**SAC**) (Haarnoja et al., 2018) which is an off-policy algorithm that optimizes a stochastic policy.

Implementation. We implemented our ARPO on the Ray RLlib framework (Liang et al., 2018) and used a CNN-based model for the policy network for these experiments. For all the experiments with ARPO, we set the $\beta_1 = \beta_2 = 20$ (in equation 3, and 5). The generator's hyperparameters are set to $\lambda_{cls} = 1$, and

Figure 4: Some snippets from Distracting control suite (Stone et al., 2021).

Table 1: [**Train Reward**] Sample efficiency results on Procgen environments. The results are the after training agents for 20M timesteps. The mean value and standard deviation are calculated across 3 random seed runs. ARPO achieves the best generalization results in many environments compared to PPO and RAD. Best agent is in **Bold**.

Env	ARPO (ours)	PPO	RAD
jumper	**8.46** ± 0.22	8.41 ± 0.07	7.94 ± 0.47
bigfish	**10.69** ± 2.26	8.19 ± 1.96	7.48 ± 0.61
climber	**7.56** ± 0.58	6.85 ± 0.92	6.23 ± 0.63
caveflyer	**6.78** ± 0.48	6.14 ± 1.05	5.12 ± 0.49
miner	**10.62** ± 0.83	7.87 ± 1.13	6.59 ± 1.16
fruitbot	**27.29** ± 0.68	27.26 ± 0.96	26.88 ± 0.35
leaper	**4.47** ± 0.35	3.98 ± 1.06	2.72 ± 0.51
ninja	8.07 ± 0.35	**8.08** ± 0.78	6.11 ± 0.82
dodgeball	**5.34** ± 0.4	5.13 ± 0.51	3.8 ± 0.31
maze	**9.09** ± 0.18	7.77 ± 0.59	6.26 ± 0.42
plunder	8.66 ± 1.05	**9.55** ± 1.03	8.14 ± 2.0
starpilot	24.61 ± 3.16	26.17 ± 1.22	**27.36** ± 3.13
heist	**5.1** ± 0.26	4.7 ± 0.44	3.93 ± 0.5
coinrun	8.54 ± 0.68	**9.57** ± 0.09	9.15 ± 0.24
bossfight	4.11 ± 1.04	6.87 ± 0.56	**7.97** ± 0.82
chaser	2.51 ± 0.17	**2.82** ± 0.55	2.5 ± 0.27

$\lambda_{rec} = 10$. Number of cluster is set to 3. Details of model configurations and parameters are given in the Appendix.

4.2 Procgen Results

Sample Efficiency. We evaluated and compared our agent on the sample efficiency during training. This result shows how quickly our agent achieves a better score during training. Table 1 shows the results after training for 20 million timesteps. Learning curve can be found in Appendix.

In many environments, we see our agent ARPO perform better than the baselines PPO and RAD. These results show that despite optimizing for the generalization, our adversarial training helps the ARPO agent learn a robust policy which is also better during training.

Generalization. The results in Figure 5 show the test results of ARPO and baselines PPO and RAD on 16 Procgen environments with image observation. Overall, ARPO performs better in many environments. We observe that, almost in half of the environments, our agent ARPO outperforms both baselines. It also performs comparably in a few other environments. Note that in some environments such as Ninja, Plunder, Chaser, Heist, all the agents fail to obtain a reasonable training accuracy (Cobbe et al., 2020) and thus perform poorly during testing (generalization). This result shows that our proposed agent ARPO achieves a strong performance and reduces the generalization gap compared to the tested baselines on generalization.

We further show how each agent achieves generalization during the entire timesteps. Consequently, we compute the reward score achieved by each agent after training for 20 million timesteps. Finally we report mean and standard deviation of these scores across 3 seed runs. The generalization results are computed by evaluating the trained agents on test levels (full distribution). The performance is computed by evaluating each agent for 10 random episode trials after a training interval of 10. Table 2 shows the results comparison for all the agents on Procgen environments. Similar to the previous discussion, we see that in many of the environments, our ARPO agents perform better than the baselines. ARPO performs better than both the baselines in many environments.

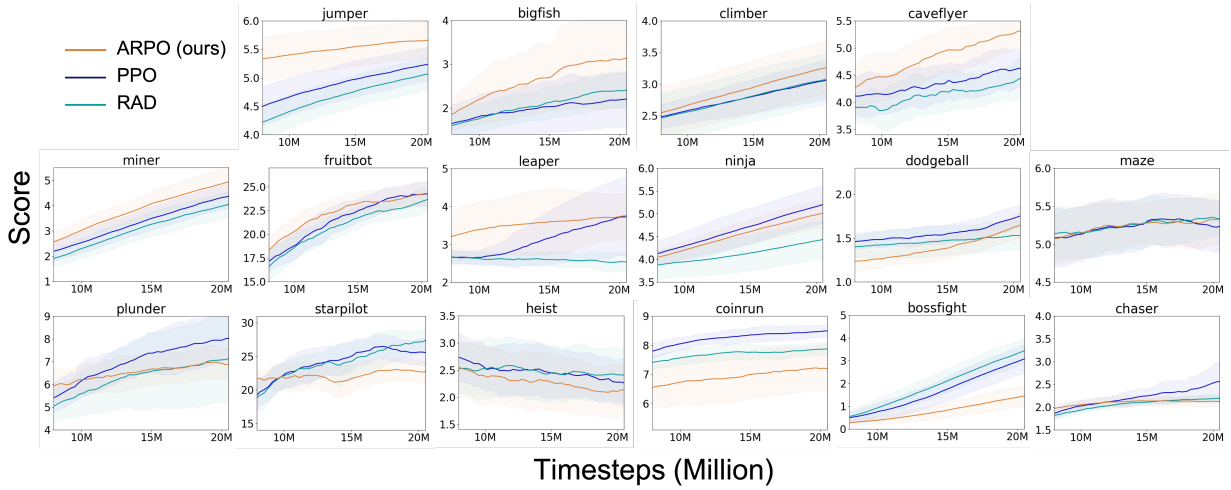


Figure 5: Generalization result learning curve. Test performance comparison on Procgen environments with image observation. ARPO performs better in many environments where it outperforms the baselines. The results are averaged over 3 seeds.

Table 2: [**Test Reward**] Generalization results on Procgen environments. The results are the after training agents for 20M timesteps. The mean value and standard deviation are calculated across 3 random seed runs. ARPO achieves the best generalization results in many environments compared to PPO and RAD. Best agent is in **Bold**.

Env	ARPO (ours)	PPO	RAD
jumper	5.65 ± 0.34	**5.96** ± 0.05	5.95 ± 0.28
bigfish	**2.92** ± 0.73	2.24 ± 0.57	2.48 ± 0.55
climber	**4.73** ± 0.47	4.37 ± 0.44	3.95 ± 0.57
caveflyer	**5.46** ± 0.21	4.77 ± 0.13	4.54 ± 0.52
miner	**6.45** ± 0.33	5.95 ± 0.38	5.4 ± 0.33
fruitbot	24.51 ± 0.29	24.4 ± 1.57	**25.02** ± 1.98
leaper	3.96 ± 0.47	**4.14** ± 1.13	2.71 ± 0.25
ninja	5.89 ± 0.47	**6.44** ± 0.59	5.06 ± 0.32
dodgeball	**2.02** ± 0.18	2.0 ± 0.03	1.68 ± 0.25
maze	5.37 ± 0.41	4.79 ± 0.35	**5.49** ± 0.03
plunder	6.38 ± 0.22	**7.88** ± 0.83	7.22 ± 2.01
starpilot	21.86 ± 0.58	26.56 ± 4.0	**27.28** ± 0.55
heist	1.9 ± 0.49	**2.18** ± 0.57	2.16 ± 0.27
coinrun	7.01 ± 0.97	**8.53** ± 0.15	7.93 ± 0.26
bossfight	2.88 ± 0.73	6.4 ± 1.3	**7.17** ± 1.1
chaser	2.18 ± 0.02	**2.85** ± 0.75	2.21 ± 0.06

4.3 Distracting Control Results

Figure 6 shows the results of ARPO and PPO on Walker walk environment from distracting control suite. We observe that ARPO performs better compared to the baseline algorithm PPO in both sample efficiency (train) and generalization (test) for background and color distraction. Note that we observe large variance (standard deviation) in the results across the run, particularly in test time. These results correspond to the benchmark results where Stone et al. (2021) also find comparatively larger variance in reported results. In some cases (Figure 6b), the test performance surpasses the corresponding timestep's train reward. This scenario might happen because the videos for train and test environments have different difficulties. Thus in

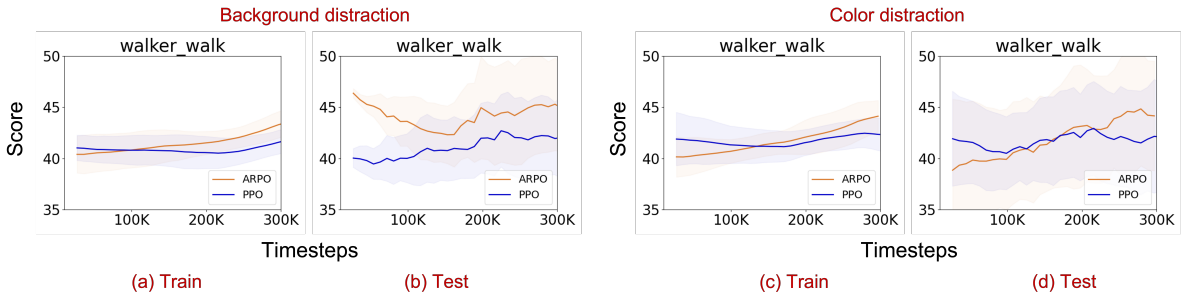


Figure 6: Sample efficiency (Train) and generalization (Test) results on Distracting control Walker walk environments. The results are averaged over 3 seeds.

some cases, the agent finds better test performance than the training. However, our ARPO's performance remains better during testing compared to PPO. Further, ARPO (and PPO) performs better compared to the off-policy SAC in both sample efficiency and generalizations in our settings. Detailed results are in Appendix.

Furthermore, results analysis for Cheetah run environment is in the Appendix.

4.4 Qualitative Analysis of Adversarial Style Transfer

In this section we discuss how the generator performs in generating adversarially translated observation during agent training. Figure 7, 8, and 9 show sample translation from the generator. Please see the corresponding caption for detailed discussion. Some additional qualitative results are in the Appendix.

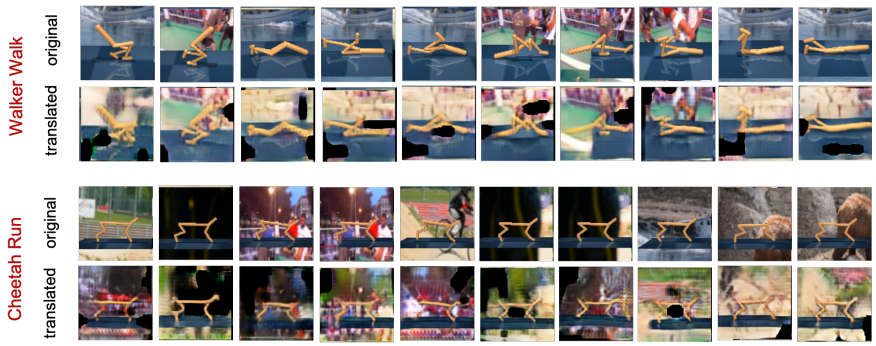


Figure 7: Sample translation of generator on **background** distraction for Walker-walk environment from Distracted Control Suite benchmark. We see that the translated observations retain the semantic that is the robot pose while changing non-essential parts. Interestingly, we observe that the adversarial generator blackout some non-essential parts of the translated images. This scenario might indicate that the min-max objective between policy network and generator tries to recover the actual state from the observation by removing parts irrelevant to the reward.

5 Related Work

Data augmentation in RL. In recent time, many approaches have been proposed to address generalization challenges including various data augmentation approaches (Cobbe et al., 2019; Laskin et al., 2020a; Kostrikov et al., 2020; Raileanu et al., 2020; Laskin et al., 2020b; Igl et al., 2019; Osband et al., 2018; Lee et al., 2020; Wang et al., 2020; Zhang et al., 2021) have shown to improve generalization. The common theme of these approaches is to increase diversity in the training data; however, these perturbations are primarily performed in isolation of the RL reward function, which might change crucial aspects of the observation, resulting in sub-optimal policy learning. In contrast, we propose a style transfer that tries to generate semantically

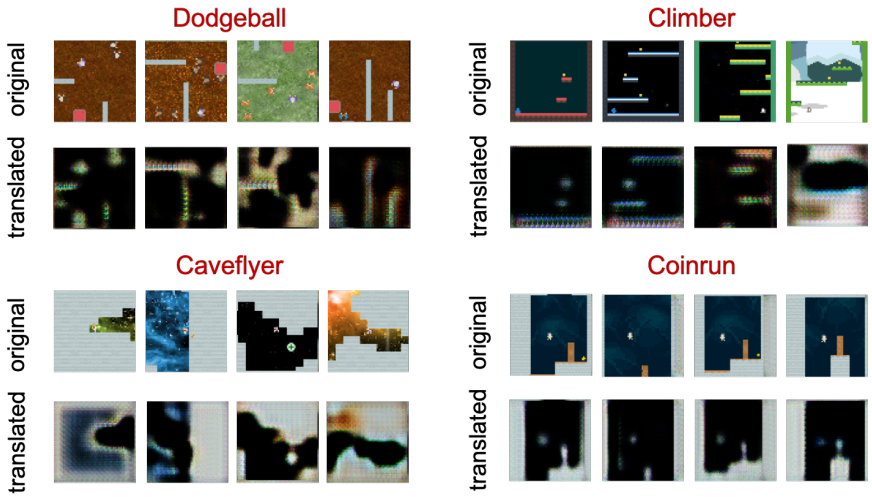


Figure 8: Sample translation of generator on four Procgen environments. We see that the generator retains most of the game semantic while changing the background color and the texture of various essential objects.

Figure 9: Sample translation of generator during various training phases on **background** distraction for Walker-walk environment from Distracted Control Suite benchmark. [**Second row**] We see that when translated back to the same cluster (in the second row), the generated images get some distortion (e.g., blackout some parts), while the essential parts remain mostly intact. This scenario might be happening due to the regularization of the cycle consistency loss by the KL component in equation 5. [**Third row**] We see that the translation to a different cluster generates images that changes the irrelevant parts while mostly keeping the semantic intact. In both the second and third row, the translation gets better as time progresses, suggesting the adaptation of both policy and generator networks due to the min-max objective.

similar but different visual style observation to train a robust policy. Furthermore, our style perturbation takes into account the reward signal from the policy.

Note that the base RL algorithm (e.g., PPO) in our ARPO method still uses the original observation from the environment while participating in adversarial objectives (see Figure 1). Thus, data augmentation can be applied to the original observations before feeding them to the base RL algorithm to potentially improve the base RL agent, as observed in many data augmentation techniques discussed above. NEW

The experiment setup of SVEA Hansen et al. (2021) is different because they assume no visual distraction during training but distraction during test time. This scenario is in contrast to the benchmark we are evaluating. In this paper, we are also evaluating the existence of distraction during training.

Style Transfer in RL. Many approaches have been proposed which use style transfer to improve reinforcement learning algorithms. Gamrian & Goldberg (2019) use image-to-image translation between domains to learn the policy in zero-shot learning and imitation learning setup. They use different levels' data and train

unpaired GANs between two levels (domains), which requires access to the annotated agent's trajectory data in both source and target domains. Furthermore, Smith et al. (2019) performs pixel-level image translation via CycleGAN to convert the human demonstration to a video of a robot. These translated data can then be used to convert the human demonstration into a video of a robot used to construct a reward function that helps learn complex tasks. However, their CycleGAN operates on two predefined domains, human demonstration and robot video.

In contrast, we first automatically generate the domains using a clustering algorithm and then train the style-translation generator. In our case, we do not need the information of levels; instead, we automatically cluster the trajectory data based on visual features of the observations. Additionally, our visual-based clustering may put observations from multiple levels into a cluster, potentially preventing GAN from overfitting Gamrian & Goldberg (2019) to any particular environment levels.

Adversarial Approach in RL. Adversarial methods have been used in the context of reinforcement learning to improve policy training. Li et al. (2021) proposes to combine data augmentation with auxiliary adversarial loss to improve generalization. In particular, they use an adversarial discriminator to predict the label of the observation. However, they need to know the domain label, limiting the applicability as it might not be available, and the number of labels might be large, for example, in procedural generation. In contrast, we use an unsupervised clustering that automatically finds out similar visual features later used for generator training. (Pinto et al., 2017b) use adversarial min-max training to learn robust policy against forces/disturbances in the system (Pinto et al., 2017b). In addition, adversarial training has been used between two robots to improve object grasping (Pinto et al., 2017a) and in the context of multi-agent self-play scenario (Heinrich & Silver, 2016). Zhang & Guo (2021) generate adversarial examples for observations in a trajectory by minimizing the cumulative reward objective of the reinforcement learning agent. In contrast to these methods, we use adversarial visual-based style-transfer of the observation to guide robust policy learning by enforcing the policy to produce similar output action distributions.

Generalization in RL. Many methods have been proposed to reduce the generalization gap between training and unseen test environment in the context of reinforcement learning. State learning approaches (Higgins et al., 2017; Agarwal et al., 2021a; Zhang et al., 2020) and auto-encoder based latent state learning (Lange et al., 2012; Lange & Riedmiller, 2010; Hafner et al., 2019) with reconstruction loss have been proposed. The sequential structure in RL (Agarwal et al., 2021a) and behavioral similarity between states (Zhang et al., 2020) to learn invariant states have been leveraged to improve RL robustness and generalization. In contrast, we learn the invariant state by providing the RL agent with an additional variant of the same observation while retaining the reward structure using an adversarial max-min objective.

Our primary focus is to demonstrate how the adversarial style transfer helps learn a robust policy and improves the robustness of the base RL policy used. Moreover, in our case, the final policy is still trained FIX using original observation. Thus this policy can be regularized using the method proposed in DRAC (Raileanu et al., 2020). The general data augmentation can still be used on observation before passing it to policy. Thus our method might be applied to many existing algorithms, including data augmentation and regularization. . Combining our method with these algorithms could improve performance; investigating them can be an interesting avenue for future work.

6 Discussion

In summary, we proposed an algorithm, Adversarial Robust Policy Optimization (ARPO), to improve generalization for reinforcement learning agents by removing the effect of overfitting in high-dimensional observation space. Our method consists of a max-min game theoretic objective where a generator is used to transfer the style of high-dimensional observation (e.g., image), thus perturb original observation while maximizing the agent's probability of taking a different action for the corresponding input observation. In contrast, a policy network updates its parameters to minimize the effect of such translation, thus being robust to the perturbation while maximizing the expected future reward. We tested our approaches on Procgen and distracting control benchmarks in generalization setup and confirmed the effectiveness of our proposed algorithm. Furthermore, empirical results show that our method generalizes better in unseen test environments than the tested baseline algorithms.

Note that the lack of visual diversity in the observation of the environment might result in poor clustering, which eventually leads to a less challenging style translation by the adversarial generator. However, in those scenarios, the reinforcement learning algorithms often perform well as the train and test environment remain consistent (e.g., same background). Furthermore, this setup might lead the agent to overfit the training environment, which results in the agent performing poorly in a slightly different environment (Song et al., 2020; Zhang et al., 2018b). In this paper, we are interested in the scenario where the train and test environment are visually different, which we think is a more practical setup (Cobbe et al., 2020; Stone et al., 2021).

References

Rishabh Agarwal, Marlos C. Machado, Pablo Samuel Castro, and Marc G. Bellemare. Contrastive behavioral similarity embeddings for generalization in reinforcement learning. In *International Conference on Learning Representations*, 2021a.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021b.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pp. 214–223. PMLR, 2017.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8789–8797, 2018.

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pp. 1282–1289. PMLR, 2019.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pp. 2048–2056. PMLR, 2020.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

Shani Gamrian and Yoav Goldberg. Transfer learning for related reinforcement learning tasks via image-to-image translation. In *International Conference on Machine Learning*, pp. 2063–2072. PMLR, 2019.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pp. 2555–2565. PMLR, 2019.

Nicklas Hansen, Hao Su, and Xiaolong Wang. Stabilizing deep q-learning with convnets and vision transformers under data augmentation. *Advances in Neural Information Processing Systems*, 34, 2021.

Moritz Hardt, Ben Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *International Conference on Machine Learning*, pp. 1225–1234. PMLR, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.

Irina Higgins, Arka Pal, Andrei A Rusu, Loic Matthey, Christopher P Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. *arXiv preprint arXiv:1707.08475*, 2017.

Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. In *Advances in neural information processing systems*, pp. 13978–13990, 2019.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. In *International Conference on Machine Learning*, pp. 1857–1865. PMLR, 2017.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*, 2021.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2010.

Sascha Lange, Martin Riedmiller, and Arne Voigtländer. Autonomous reinforcement learning on raw visual input data in a real world application. In *The 2012 international joint conference on neural networks (IJCNN)*, pp. 1–8. IEEE, 2012.

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. In *Advances in neural information processing systems*, 2020a.

Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pp. 5639–5650. PMLR, 2020b.

Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=HJgcvJBFvB`.

Bonnie Li, Vincent François-Lavet, Thang Doan, and Joelle Pineau. Domain adversarial reinforcement learning. *arXiv preprint arXiv:2102.07097*, 2021.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pp. 3053–3062. PMLR, 2018.

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

Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720*, 2018.

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 8617–8629, 2018.

Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*, 2018.

Lerrel Pinto, James Davidson, and Abhinav Gupta. Supervision via competition: Robot adversaries for learning tasks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1601–1608. IEEE, 2017a.

Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *International Conference on Machine Learning*, pp. 2817–2826. PMLR, 2017b.

Roberta Raileanu, Max Goldstein, Denis Yarats, Ilya Kostrikov, and Rob Fergus. Automatic data augmentation for generalization in deep reinforcement learning. *arXiv preprint arXiv:2006.12862*, 2020.

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

Laura Smith, Nikita Dhawan, Marvin Zhang, Pieter Abbeel, and Sergey Levine. Avid: Learning multi-stage tasks via pixel-level translation of human videos. *arXiv preprint arXiv:1912.04443*, 2019.

Xingyou Song, Yiding Jiang, Stephen Tu, Yilun Du, and Behnam Neyshabur. Observational overfitting in reinforcement learning. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=HJli2hNKDH`.

Austin Stone, Oscar Ramirez, Kurt Konolige, and Rico Jonschkowski. The distracting control suite – a challenging benchmark for reinforcement learning from pixels. *arXiv preprint arXiv:2101.02722*, 2021.

Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. dm_control: Software and tasks for continuous control, 2020.

Huan Wang, Stephan Zheng, Caiming Xiong, and Richard Socher. On the generalization gap in reparameterizable reinforcement learning. In *International Conference on Machine Learning*, pp. 6648–6658. PMLR, 2019.

Kaixin Wang, Bingyi Kang, Jie Shao, and Jiashi Feng. Improving generalization in reinforcement learning with mixture regularization. *arXiv preprint arXiv:2010.10814*, 2020.

Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018a.

Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020.

Amy Zhang, Rowan Thomas McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=-2FCwDKRREu`.

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

Hanping Zhang and Yuhong Guo. Generalization of reinforcement learning with policy-aware adversarial data augmentation. *arXiv preprint arXiv:2106.15587*, 2021.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232, 2017.

Appendix

A Additional Procgen Results

Figure 10 shows the sample efficiency results for ARPO, PPO, and RAD.

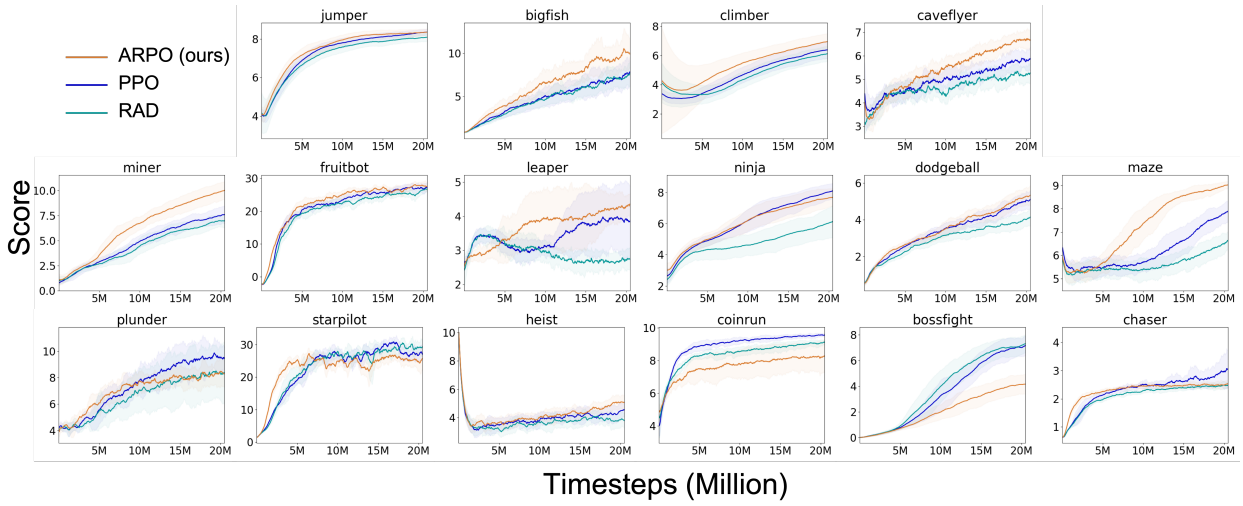


Figure 10: Sample efficiency results. Train performance (learning curve) comparison on Procgen environments. Despite using perturbation on the observation, our agent ARPO outperforms the baselines in many environments, thus achieving better sample efficiency during training. The results are averaged over 3 seeds.

B Additional Distracting Control Results

Comparison with SAC on Walker Walk. Results comparison on Distracting Control for ARPO, PPO, and SAC are in Figure 11 (Walker walk).We see that ARPO (and PPO) perform better compared to the SAC in both sample efficiency (train) and generalization (test) in these settings.

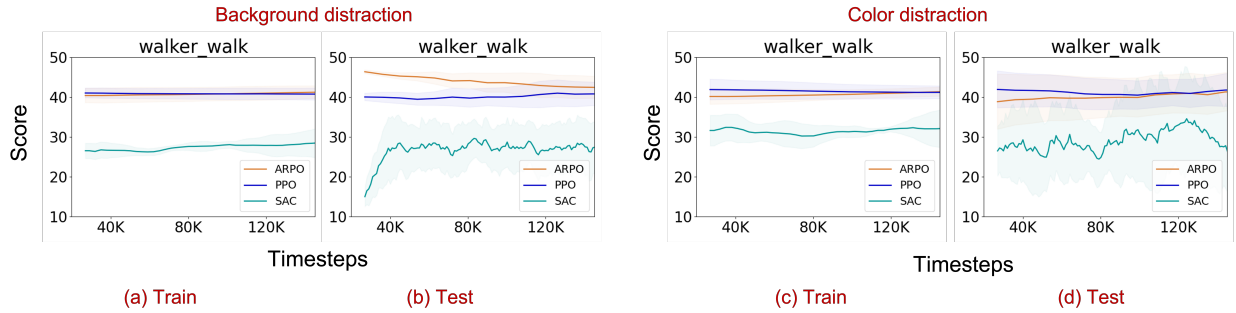


Figure 11: Distracting control Walker walk environment results (including SAC). The results are averaged over 3 seeds. **(a, c)** Sample efficiency (train) **(b, d)** Generalization (test).

Results on Cheetah run with background distraction.

Results for ARPO and PPO on Cheetah run with background distraction are in Figure 12. We see that both agents perform comparably in both background and color distraction. We observe similar results for both the agents in this setup of Cheetah run environments. Note that in this training timestep the PPO agents perform poorly in the training and testing. The highest achievable reward for this environment is much higher (see Stone et al. (2021)). Consequently the the ARPO agent could not improve over the base

algorithm PPO. This suggest that the base algorithm needs to trained sufficiently to achieve any reasonable reward before we benefit of the adversarial training.

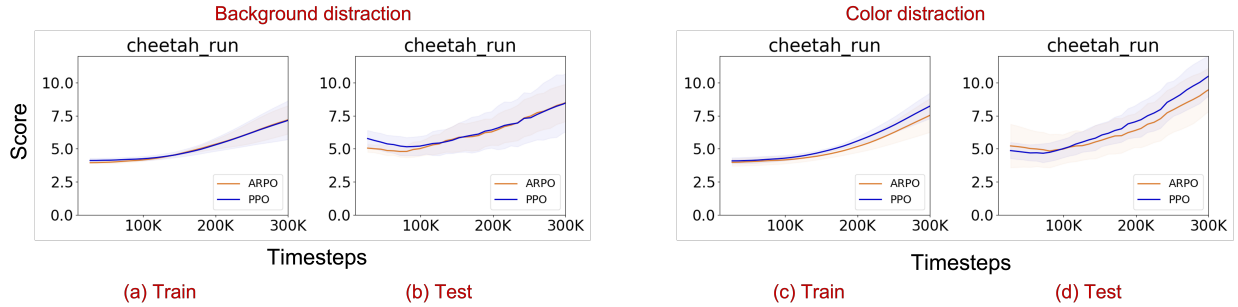


Figure 12: Distracting control Cheetah run environment results. The results are averaged over 3 seeds. **(a, c)** Sample efficiency (train), and **(b, d)** Generalization (test).

Note that the resulting score is low compared to the reported results in the benchmark paper Stone et al. (2021) which suggests the PPO itself could not learn helpful behavior in this environment.

Comparison with SAC on Cheetah Run Results comparison on Distracting Control for ARPO, PPO, and SAC on Cheetah run are in Figure 13.

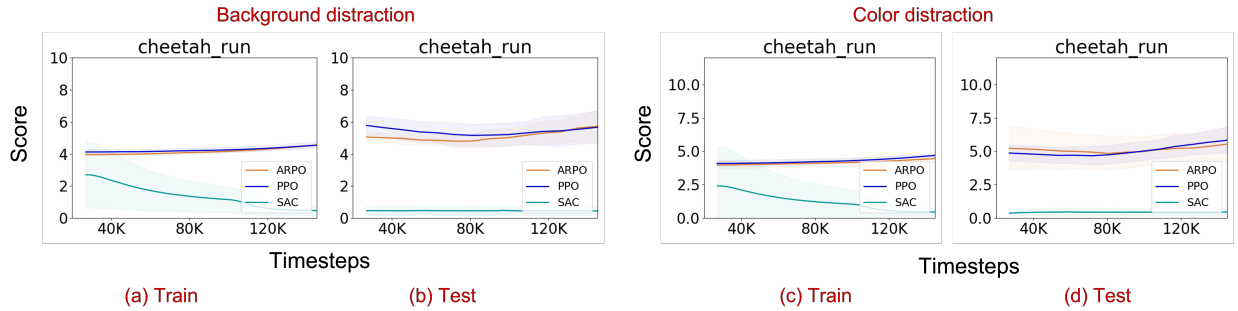


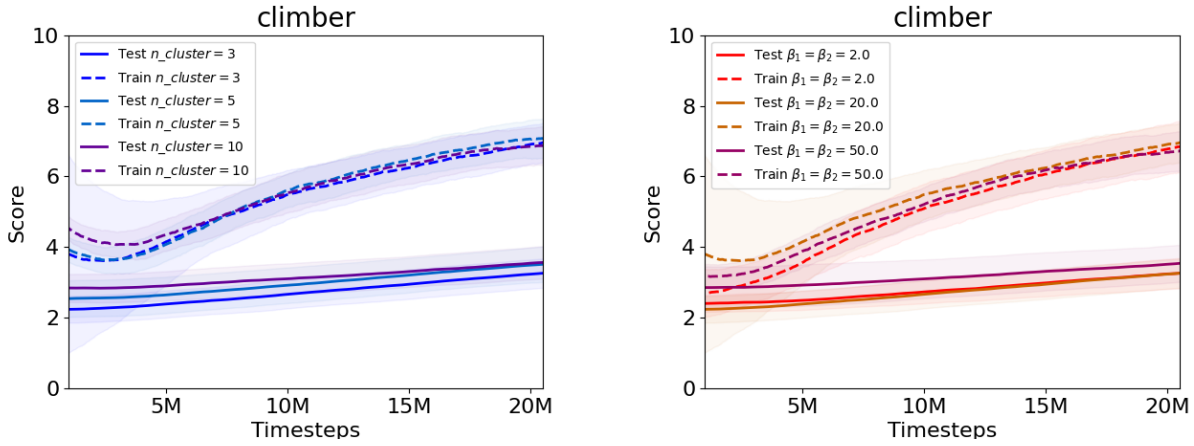
Figure 13: Distracting control Cheetah run environment results (including SAC). The results are averaged over 3 seeds. **(a, c)** Sample efficiency (train), and **(b, d)** Generalization (test).

C Ablation Study and Hyperparameters

C.1 Hyperparameters of ARPO

We conduct a study on how ARPO agent's performance varies due to its hyperparameters: (a) the number of clusters generated by the GMM model for generator training, and (b) β_1, β_2 which indicate the amount of participation of policy and generator in the adversarial objective.

Figure 14 shows the ablation results. We observe that our ARPO agent shows improvement in generalization (test) performance with the increase in number of cluster on Climber environment (Figure 14a). As the number of cluster increase, the generator has more option to choose for translation, and thus the policy might face a hard challenge and thus learn a more robust policy. In this paper, we report the ARPO agent's results on cluster $n\_cluster = 3$. On the other hand, and ablation results on different β values are shown in Figure 14b. When β values are large the test performance goes up; however, training performance suffer a bit. In this paper, we report the results on $\beta_1 = \beta_2 = 20$, which shows a balance in train and test performance.



(a) (b)

Figure 14: The results are averaged over 3 seeds. **(a)** Our ARPO agent's results on different cluster numbers. It improves generalization (test) performance with the increase in cluster number in the Climber environment. **(b)** Our ARPO agent's results on different β values which determine participation on adversarial optimization.

C.2 KL Regularization

Figure 15 shows the KL regularization results in different policy iteration on four Procgen environments. For each environment, we run the experiment and collect KL value $(KL(\pi_\theta(.|x_t), \pi_\theta(.|x'_t)))$ for each policy iteration. To calculate moving average, we select a window size of 500 and the standard deviation is calculated on these 500 values, and showed in the shaded area in Figure 15. Results analysis is in the caption. FIX

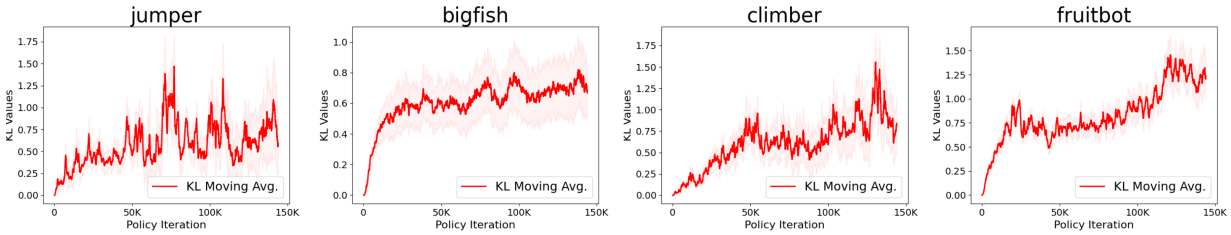


Figure 15: KL regularization during RL training for four Procgen environments. At the beginning of the training, both the policy and generator behave randomly or outputting same action for all the images; thus, the KL values become nearly zero. Gradually, the KL started to increase as the policy learning progressed with the adversarial generated observation. Eventually, the policy tries to adjust to the changes in the observation, and the KL values become stable (**bigfish**) or going downward (**jumper**, and **climber**). We see that in **fruitbot** environment, the values started to increase at the end, which suggests a possible divergence. This scenario might be an explanation of why the generalization performance started to drop at the end in Figure 5, possible overfitting. Thus this KL measure might be a tool to detect possible divergence and overfitting. However, each environment's stability cutoff time (policy iteration) is different, suggesting that the KL regularization behavior might be different depending on the environment complexity. Ideally, with enough training, the KL values should converge to zero (again) if the policy reaches optimal (for true MDP state) and the generator fully recovers the true state from the observation and changes all the irrelevant information from the observation. Note that we limit the policy to train until a cutoff time (20 Million RL timesteps).

D Additional Qualitative Results

Figure 16 shows sample translation from the generator on color distracted environments.

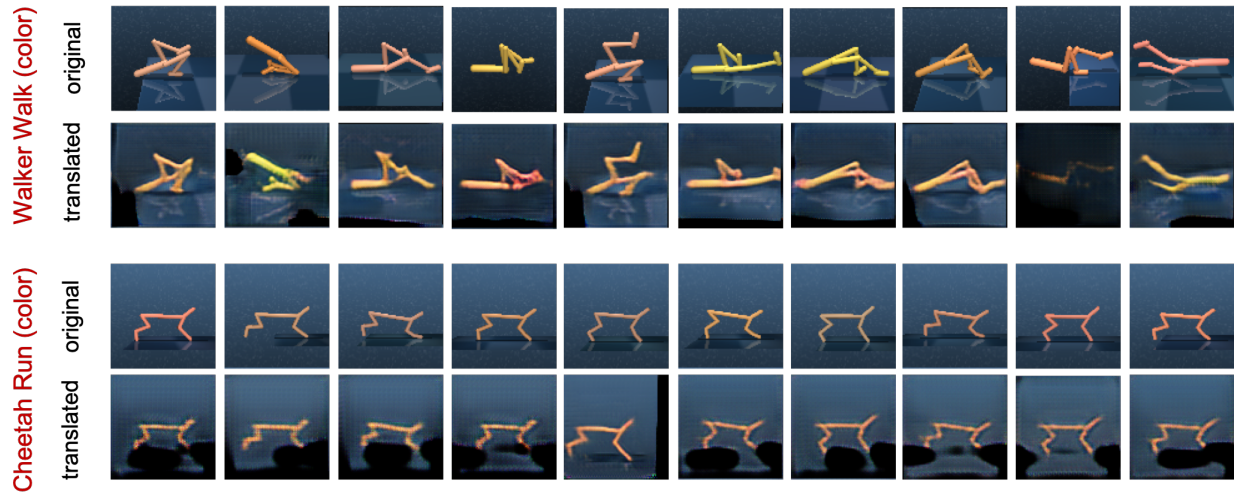


Figure 16: Sample translation of generator on **color** distraction for Walker-walk and Cheetah-run environments from Distracted Control Suite benchmark.

E Training Loss of StarGAN Generator

The adversarial loss is calculated as the Wasserstein GAN (Arjovsky et al., 2017) objective with gradient penalty which is defined as

$$\begin{aligned}\mathcal{L}_{adv} = \mathbb{E}_x[D_{src}] - \mathbb{E}_{x,c}[D_{src}(G(x,c))] \\ -\lambda_{gp}\mathbb{E}_{\hat{x}}[(||\nabla_{\hat{x}} D_{src}(\hat{x})|| - 1)^2],\end{aligned} \tag{6}$$

where $\hat{x}$ is sampled uniformly along a straight line between a pair of real and generated fake images. Here, the gradient penalty helps to stabilize the GAN training process and has been shown to perform better than traditional GAN training (Choi et al., 2018; Arjovsky et al., 2017; Gulrajani et al., 2017). We set the hyperparameter $\lambda_{gp} = 10$.

The classification loss of real image is defined as

$$\mathcal{L}^r_{cls} = \mathbb{E}_{x,c'}[-\log D_{cls}(c'|x)], \tag{7}$$

where $D_{cls}(c'|x)$ is the probability distribution over all clusteer labels. Similarly, the classification loss of fake generated image is defined as

$$\mathcal{L}^f_{cls} = \mathbb{E}_{x,c}[-\log D_{cls}(c|G(x,c))], \tag{8}$$

The reconstruction loss with the generator objective defined as

$$\mathcal{L}_{rec} = \mathbb{E}_{x,c,c'}[||x - G(G(x,c),c')||_1], \tag{9}$$

where we use the $L1$ norm.

F Environment Details

Procgen We conducted experiments on OpenAI Procgen Cobbe et al. (2020) benchmark, consisting of diverse procedurally-generated environments with different action sets. This environment has been used

Table 3: Hyperparameters for Procgen (RLlib) Experiments

Description	Hyperparameters
Discount factor of the MDP	$gamma$: 0.999
The GAE(lambda) parameter	$lambda$: 0.95
The default learning rate	lr : $5.0e-4$
Number of epochs per train batch	num_sgd_iter : 3
Total SGD batch size	$sgd\_minibatch\_size$: 2048
Training batch size	$train\_batch\_size$: 16384
Initial coefficient for KL divergence	kl_coeff : 0.0
Target value for KL divergence	kl_target : 0.01
Coefficient of the value function loss	vf_loss_coeff : 0.5
Coefficient of the entropy regularizer	$entropy\_coeff$: 0.01
PPO clip parameter	$clip\_param$: 0.2
Clip param for the value function	vf_clip_param : 0.2
Clip the global amount	$grad\_clip$: 0.5
Default preprocessors	$deepmind$
PyTorch Framework	$framework$: $torch$
Settings for Model	$custom\_model$: $impala\_cnn\_torch$
Rollout Fragment	$rollout\_fragment\_length$: 256

to measure how quickly (sample efficiency) a reinforcement learning agent learns generalizable skills. These environments greatly benefit from the use of procedural content generation, the algorithmic creation of a near-infinite supply of highly randomized content. The design principles consist of high diversity, tunable difficulty, shared action, shared observation space, and tunable dependence on exploration. Procedural generation logic directs the level layout and other game-specific details. Thus, to master any of these environments, agents must learn an effective policy across all environment variations. We use all 16 environments available in this benchmarks.

All environments use a discrete 15 dimensional action space which generates $64 \times 64 \times 3$ RGB image observations. Note that some environments may use no-op actions to accommodate a smaller subset of actions.

G Implementation Details

Procgen Experiments For experimenting on Procgen environments, we used RLlib Liang et al. (2018) to implement our ARPO, and baselines PPO and RAD cutour color algorithms. For all the agents' policy network (model), we use a CNN architecture used in IMPALA Espeholt et al. (2018) which is the best performing model in Procgen benchmark (Cobbe et al., 2020). We use the same policy parameters for all agents for a fair comparison.

Policy learning hyperparameter settings for all the agents (ARPO, PPO, and RAD) are set same for fair comparison and detect the effect of our proposed method (max-min adversarial objective with perturbation network). These hyperparameters are given in Table 3. Note that only the custom parameters are given here, other defaults parameter values can be found in the RLlib library Liang et al. (2018).

Distracting Control Experiments For experimenting on Distracting Control Suite (Stone et al., 2021) environments, we used RLlib Liang et al. (2018) to implement our ARPO, and baselines PPO and SAC algorithms. We use the PPO's CNN-based policy network (model) from RLlib for ARPO, and PPO. The policy specific parameters for ARPO, and PPO are the same which are given in Table 4.

For SAC, we use it's CNN-based model available in RLlib. The policy specific parameters are given in Table 5. Note that only the custom parameters for the RLlib implementation are given here, other defaults parameter values can be found in the RLlib library Liang et al. (2018).

Computing details. We used the following machine configuration to run our experiments: 20 core-CPU with 256 GB of RAM, CPU Model Name: Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz, and a Nvidia A100 GPU. NEW

Table 4: ARPO, and PPO Hyperparameters for Distracting Control Experiments

Description	Hyperparameters
Number of epochs per train batch	num_sgd_iter : 3
Total SGD batch size	$sgd\_minibatch\_size$: 256
Training batch size	$train\_batch\_size$: 8192
PyTorch Framework	$framework$: $torch$

Table 5: SAC Hyperparameters for Distracting Control Experiments

Description	Hyperparameters
Training batch size	$train\_batch\_size$: 512
Timesteps per iteration	$timesteps\_per\_iteration$: 1000
Timesteps per iteration	$learning\_starts$: 5000
PyTorch Framework	$framework$: $torch$

H Comparison with DRAC

Figure 17 shows comparison with DRAC Raileanu et al. (2020) on Procgen Maze environment. We see that our method ARPO performs better in sample efficiency during training compared to DRAC. The results are averaged over 2 random seed runs. We further show final results after training agents for 14M timesteps.

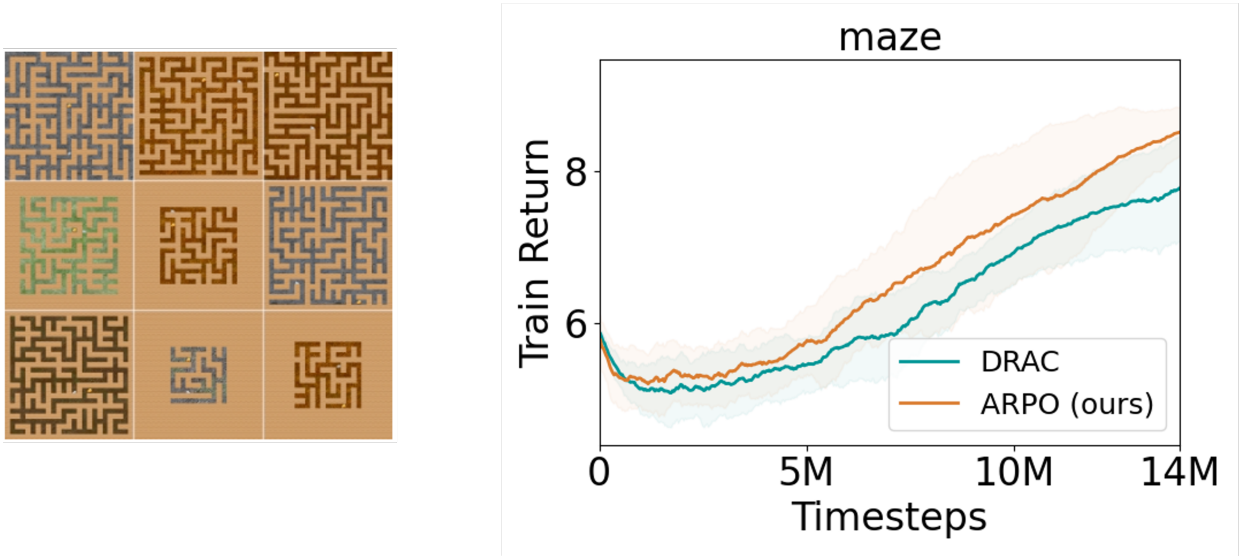


Figure 17: Training curve comparison with DRAC on Procgen Maze environment.

Table 6 shows mean and standard deviation of results. For both the agents, the returns are smoothed using standard exponential averaging to account for the performance in previous timesteps. We see that our agent ARPO performs better in final results in train return compared to DRAC. On the other hand, both agents show similar mean values for test return. However, our method ARPO shows a smaller variance (0.24) compared to DRAC's high variance (1.38). These results demonstrate that overall our ARPO algorithm shows effective results compared to DRAC in the Procgen Maze environment. Note that in this setting DRAC uses tuned data augmentation (see the DRAC paper Raileanu et al. (2020)) which is *crop* in case of maze. On the other hand, our method does not assume any such set of data augmentation. Instead, our method tries to figure out such perturbation on observation as the training progresses.

Setup. We implemented the DRAC method following the original paper Raileanu et al. (2020). In particular, we incorporated policy and value function regularization following the implementation details in the original DRAC implementation. For a fair comparison, we keep all other hyperparameters the same as we did for all other experiments. NEW

Table 6: Results on Maze after 14M timesteps.

Eval	ARPO (ours)	DRAC
Train	**8.55± 0.29**	7.82± 0.66
Test	**5.24 ± 0.24**	5.22± 1.38

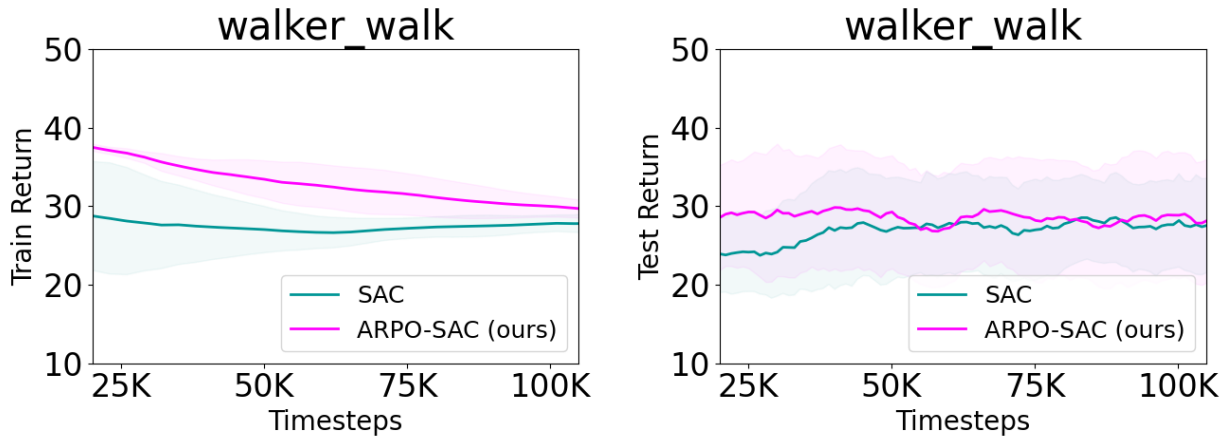


Figure 18: **Results comparison of ARPO on SAC agent in Distracting control Walker walk environment with background distraction. Our agent ARPO with SAC base algorithm performs better in training in terms of train return. In test return, ARPO achieves slightly better generalization at the early timesteps and is similar in the rest of the timesteps.**

I ARPO on SAC

In this experiment, we investigate the use of our technique on a different RL algorithm than PPO. We choose off-policy SAC to demonstrate the implication of our method. Following a similar setup as the PPO base algorithm, we regularize the policy using the KL term in Equation 3. The RL part and generator train together alternately, similar to the PPO base setup. We name this version of our agent **ARPO-SAC**.

Figure 18 shows the results comparison on walker walk background distraction environment. Our agent ARPO-SAC shows better sample efficiency than SAC only baseline. It also shows slightly better generalization compared to SAC.

Note that PPO is an on-policy algorithm that directly learns the policy function. In contrast, SAC is an off-policy method that learns Q-functions and then generates policy from them. Thus, we see that policy gradient-based algorithms such as PPO better more amenable to being used with the adversarial setup. However, for the off-policy-based method such as SAC, Q-functions learning can be regularized with the KL-term, which might show an even better performance boost. However, investigating them in detail can be interesting future work. NEW

J DCS Results with *rliable* Metric

We use the probabilistic algorithm comparison metric from rliable Agarwal et al. (2021b). Figure 19 compares ARPO and PPO on Walker walk DCS environments. In particular, we use two environment setups: walker-walk color distraction and walker-walk background video distraction. The probability in Figure 19 gives a range of probability values based on the performance of these algorithms in several random seeds. In this experiment, we use 3 random seed runs for each agent in each environment. Based on this metric, we see

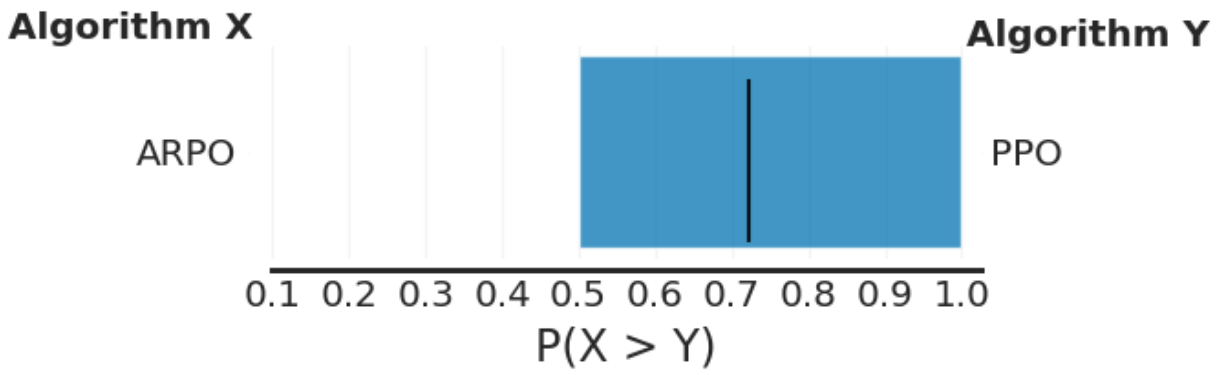
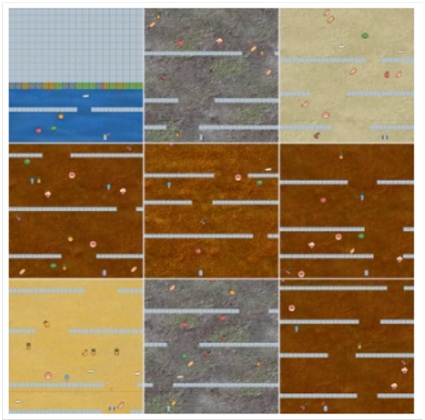


Figure 19: Probability comparison in walker-walk DCS for two detractors: Background video and color. We see that ARPO strongly outperforms PPO with a high probability, a chance of 50-100%.

that ARPO strongly outperforms PPO with a high probability, a chance from 50-100%. Overall, the results show ARPO improves performance over PPO in this setup. NEW

K Visual diversity in training environments

We conducted experiments to investigate visual diversity in training environments. Figure 20 shows the results on Procgen Fruitbot environment. We increase the training environment by allowing the agent to have more training levels. We choose Fruitbot due to its visual diversity (e.g., various background colors). For this setup, we choose different train levels 100, 200, and 500 during training and test on a fixed full distribution. We follow this use of full distribution as the test in the original Procgen benchmark papers Cobbe et al. (2020) and many baselines evaluated on this benchmark Raileanu et al. (2020); Laskin et al. (2020a). Despite similar training performance, we observe that the test performance improved for 500 levels compared to 200 and 100. Furthermore, the number of train levels 200 performs slightly better than 100



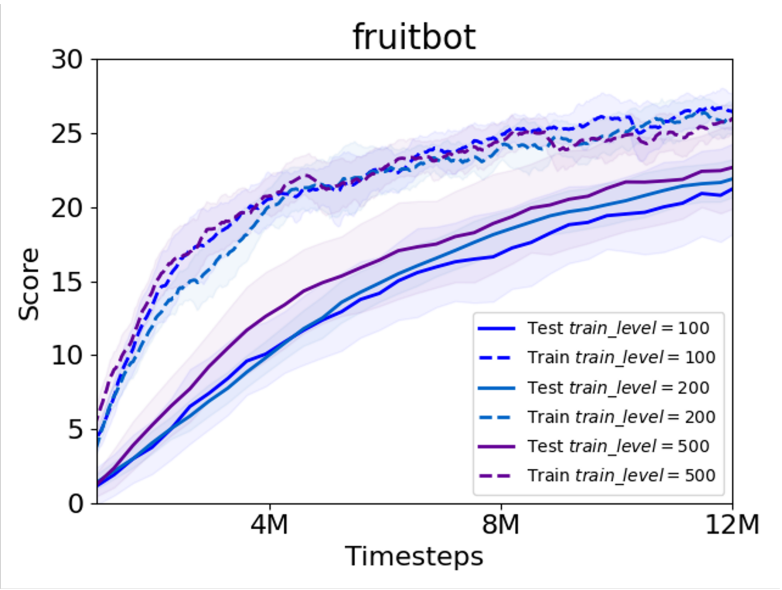


Figure 20: [**Left**] Some snippets of Procgen Fruitbot environmen. [**Right**] Performance comparison of various train level.

level in test score. However, the train results remain similar for all the agents. Note that one potential reason for improving the test performance (as observed in Procgen paper Cobbe et al. (2020)). However,

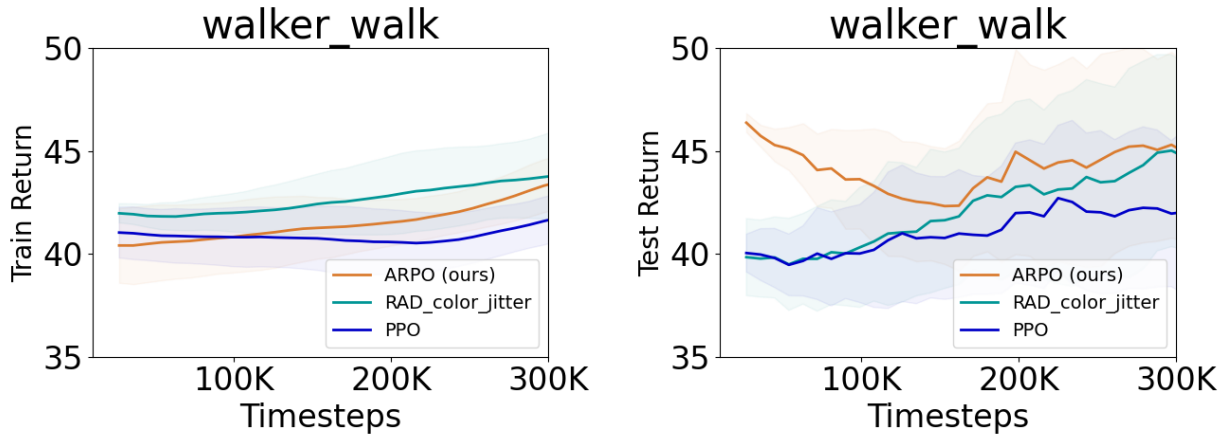


Figure 21: **Comparison of our ARPO with RAD color jitter data augmentation. We see that color jitter augmentation performs slightly better in train return. However, in generalization (test return), our ARPO agent performs better, especially in the initial timesteps.**

here we observe no such change in performance during training. Thus, the performance improvement might be due to diverge observation styles during training for our ARPO agent. NEW

L Comparison with color jitter data augmentation

Figure 21 shows results comparing our ARPO agent with color jitter data augmentation on a walker-walk background distraction environment. We follow RAD Laskin et al. (2020a) approach to use the data augmentation. In particular, the observation is augmented using color jitter and then pass it to the agent. The results are averaged over 3 random seed runs. For a fair comparison, all the RL, environment, and hyperparameters are kept the same as PPO and ARPO.