ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240874

Survival of the Fittest: Evolutionary Adaptation of Policies for Environmental Shifts

Sheryl Paul[a,*] **and Jyotirmoy V. Deshmukh**[a]

[a]University of Southern California

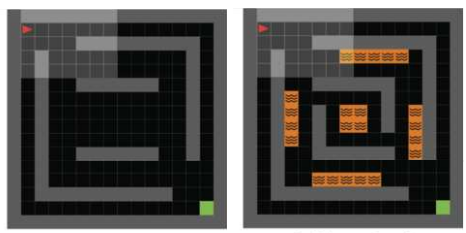
Abstract. Reinforcement learning (RL) has been successfully applied to solve the problem of finding obstacle-free paths for autonomous agents operating in stochastic and uncertain environments. However, when the underlying stochastic dynamics of the environment experiences drastic distribution shifts, the optimal policy obtained in the trained environment may be sub-optimal or may entirely fail in helping find goal-reaching paths for the agent. Approaches like domain randomization and robust RL can provide robust policies, but typically assume minor (bounded) distribution shifts. For substantial distribution shifts, retraining (either with a warm-start policy or from scratch) is an alternative approach. In this paper, we develop a novel approach called *Evolutionary Robust Policy Optimization* (ERPO), an adaptive re-training algorithm inspired by evolutionary game theory (EGT). ERPO learns an optimal policy for the shifted environment iteratively using a temperature parameter that controls the trade off between exploration and adherence to the old optimal policy. The policy update itself is an instantiation of the replicator dynamics used in EGT. We show that under fairly common sparsity assumptions on rewards in such environments, ERPO converges to the optimal policy in the shifted environment. We empirically demonstrate that for path finding tasks in a number of environments, ERPO outperforms several popular RL and deep RL algorithms (PPO, A3C, DQN) in many scenarios and popular environments. This includes scenarios where the RL algorithms are allowed to train from scratch in the new environment, when they are retrained on the new environment, or when they are used in conjunction with domain randomization. ERPO shows faster policy adaptation, higher average rewards, and reduced computational costs in policy adaptation.

1 Introduction

A significant challenge for autonomous robotic agents used in automated warehouses, autonomous driving, and multi-UAV missions is the problem of identifying the optimal motion policy, i.e., for each state in the environment, deciding the action that the agent should execute. There are several computationally efficient approaches for planning the agent's actions in deterministic and stochastic environments, especially when a model of the environment is available [14, 7, 33, 3, 13, 12, 17]. However, such models may not be available for agents deployed in highly uncertain and dynamic environments [35, 8]. Model-free reinforcement learning (RL) algorithms [29, 4] have been highly effective at learning optimal policies when the environment dynamics are unknown.

* Corresponding Author. Email: sherylpa@usc.edu

Traditional RL methods suffer from their lack of generalizability when exposed to new, unanticipated changes in the environment. Typically, these RL approaches demonstrate only moderate resistance to noise and exhibit poor performance when deployed in environments significantly different from those encountered during training. The lack of robust adaptation capabilities in these algorithms is a critical drawback, especially in applications where reliability and consistency across varied operational conditions are paramount.



Original Environment Post distribution shift

Figure 1: The left figure is an example of an original environment where the agent (red triangle) has to reach the green goal square. The right figure represents the same environment after a distribution shift, introducing additional walls and 'lava' that complicate navigation and makes it differ significantly from the original layout.

Related Work: In response to these challenges, several techniques have been developed to enhance the robustness of RL algorithms including domain randomization [21] and distributionally robust reinforcement learning ([27] [22]). Domain randomization trains models across a wide range of simulated variations, thereby improving the algorithm's immunity to noise and its performance under environmental changes. However, this method generally does not perform well if the changes to the environment are substantial.

Adversarial RL [30] and robust reinforcement learning techniques, including approaches like Monotonic Robust Policy Optimization (MRPO), specifically aim to optimize the algorithm's performance in the worst-case scenarios. These approaches involve training under conditions that include adversarial disturbances or significant noise, thereby preparing the model to handle extreme situations [34, 10]. Although these methods significantly enhance the model's resilience, they sometimes fail to provide optimal solutions in less challenging or more typical scenarios indicating a trade-off between general robustness and peak performance

Current approaches in Robust RL ([22]) focus on enabling model adaptation to bridge the gap between simulation and real-world applications. Simulation models are generally simplistic and fail to consider environmental variables such as resistance, friction, and various

other minor disturbances, so they cannot be directly deployed in risk-averse applications [11].

There is also theoretical work in developing versions of DQN such as DQN-Uncertain Robust Bellman Equation ([6]) that focuses on developing Robust Markov Decision Processes (RMDPs) with a Bayesian approach. Moreover, control theory-inspired approaches train models on subsets of underperforming trajectories. These methods focus on developing policies that exhibit greater durability and resilience in adverse conditions, often by considering worst-case performance guarantees as essential benchmarks for reliability [24], but once again, they suffer from being sub-optimal in many cases.

Approaches in transfer learning with deep reinforcement learning (Deep RL) [36] focuses on leveraging knowledge from previously learned tasks to accelerate learning in new, but related, environments. While this approach promises to improve adaptability and reduce training time, its shortcomings include difficulties in identifying which parts of knowledge are transferable and the tendency to overfit to source tasks.

So we see that despite these advancements, there remains a substantial gap in effectively adapting pre-trained models to environments that undergo significant and sudden changes. Traditional RL methods are often ill-equipped to handle situations such as alterations in factory floor layouts, unexpected blockages in warehouse paths, or disruptions in road networks due to natural disasters or construction. These scenarios can drastically alter the dynamics of the environment, rendering previous optimal actions ineffective or suboptimal, and thereby demanding either complete retraining of the models or significant adjustments to their parameters and training protocols.

Contributions. To overcome prevalent limitations in traditional reinforcement learning, we introduce a novel method that synergizes RL-based planning with principles from evolutionary game theory. We generate batches of trajectories in a simulated perturbed environment and strategically explore this environment by employing an weighted version of the policy optimal in the original setting, enhancing adaptability. Subsequently, we refine the policy by prioritizing state-action pairs that demonstrate high fitness or returns, drawing on the concept of *replicator dynamics* [20]. This evolutionary game theory concept has been successfully applied in analyzing both normative and descriptive behaviors among agents [32, 9]. Unlike traditional methods, our approach incrementally modifies the policy with new batches of data without relying on gradient calculations, ensuring *convergence to optimality with theoretical guarantees.*

We evaluate our algorithm 'Evolutionary Robust Policy Optimization' (ERPO) based on this policy update against leading deep reinforcement learning methods, including Proximal Policy Optimization (PPO) [26], PPO with Domain Randomization (PPO-DR) [21], Deep Q-Network (DQN) [15], and Advantage Actor Critic (A2C) [16], both trained from scratch and retrained from a baseline model trained on the original environment environment (denoted as PPO-B, DQN-B, and A2C-B respectively). Our method demonstrates superior performance in various standard gym environments typical in RL research. Focused on discrete state and action spaces, we have applied our model to complex versions of environments such as [31] *FrozenLake*, *Taxi*, *CliffWalking*, *Minigrid: DistributionShift*, and a challenging *Minigrid* setup featuring walls and lava (*Walls&Lava*). Our findings reveal that our algorithm not only reduces computation times but also decreases the number of training episodes required to reach performance levels comparable to or better than those achieved by the aforementioned mainstream methodologies.

2 Preliminaries

2.1 Reinforcement Learning

We model the system consisting of an autonomous agent interacting with its environment as a Markov Decision Process (MDP) defined as the tuple: $(S, A, R, \Delta, \gamma)$. At each time-step t, we assume that the agent is in some state $s_t \in S$, executes an action $a_t \in A$, transitioning to the next state $s_{t+1} \in S$, and receiving a reward $r_t \in R$ with a discount factor $\gamma \in (0, 1]$. The transition dynamics Δ is a probability of observing the next state s_{t+1} and getting the reward r_t, given that the agent is in state s_t and takes the action a_t.

In this paper, we consider finite time-horizon (denoted as T) problems under a *stochastic* policy π, a probability distribution over actions given states, such that the action a_t is sampled from the distribution $\pi(a \mid s = s_t)$ at any time t. We define $F \subseteq S$ as the set of goal states in which the agent's task is considered to be achieved. Now we formalize the sparse reward setting: if $r(s_t, a_t, s_{t+1})$ is the reward received after taking action a_t in state s_t.

$$r(s_t, a_t, s'_{t+1}) \gg r(s_t, a_t, s_{t+1}).$$

where $s'_{t+1} \in F,\ \forall s_{t+1} \notin F$.

A trajectory τ of the agent induced by policy π is defined as a $(T+1)$-length sequence of state-action pairs:

$$\tau = \{(s_0, a_0), (s_1, a_1), \ldots, (s_{T-1}, a_{T-1}), s_T\},$$
$$\text{where, } \forall t < T : a_t \sim \pi(a \mid s = s_t), (s_{t+1}) \sim \Delta(s_{t+1} \mid s_t, a_t). \tag{1}$$

We also donate a trajectory τ where actions have been sampled using the policy π as $\tau \sim \pi$. Given a trajectory, the total discounted reward of a trajectory is:

$$G^{\pi}(\tau) = \sum_{t=0}^{T} \gamma^t r_t$$

We define the state-value and action-value functions as follows:

$$v^{\pi}(s) = \mathbb{E}_{\tau \sim \pi}\left[G^{\pi}(\tau) | s_0 = s\right]$$

and

$$q^{\pi}(s, a) = \mathbb{E}_{\tau \sim \pi}\left[G^{\pi}(\tau) | s_0 = s, a_0 = a\right]$$

Let $\eta(\pi)$ be the expected discounted return for an agent under the policy π across all trajectories

$$\eta(\pi) = \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{T-1} \gamma^t r_{t+1}\right]$$

The optimal policy $\pi^{\star}$ for the MDP can then be defined as:

$$\pi^{\star} = \arg\max_{\pi}\ \eta(\pi),$$

2.2 Evolutionary Game Theory

EGT originated as the application of game-theoretic concepts to biological settings. This concept stems from the understanding that frequency-dependent fitness introduces a strategic dimension to the process of evolution [2]. It models Darwinian competition and can be a dynamic alternative to traditional game theory that also obviates the need for assumptions of rationality from the participating members. It has been applied to modeling complex adaptive systems where strategies evolve over time as those yielding higher payoffs become more prevalent, akin to the survival of the fittest in natural selection. We aim to leverage the principles of EGT [28], [25] to build an approach for our distribution shift problem.

2.2.1 Replicator Dynamics Equation:

The key concept within EGT pertinent to us is that of *replicator dynamics*, which describes how the frequency of strategies (or policies in RL) changes over time based on their relative performance. The classic replicator equation in evolutionary game theory describes how the proportion of a population adopting a certain strategy evolves over time. Mathematically, it is expressed as [18]:

$$x_j(i+1) = x_j(i)\frac{f_j(i)}{\bar{f}(i)} \tag{2}$$

where $x_j(i)$ represents the proportion of the population using strategy j at time i, $f_j(i)$ is the fitness of strategy j, and $\bar{f}(i)$ is the average fitness of all strategies at time i. The equation indicates that the growth rate of a strategy's proportion is proportional to how much its fitness exceeds the average fitness, leading to an increase in the frequency of strategies that perform better than average.

Problem Definition: We assume that we have an optimal policy for the original environment dynamics Δ, referred to as $\pi^\star_\Delta$.

Suppose that we have a new environment with dynamics Δ_{new} obtained by significantly perturbing the distribution representing Δ,[1] then the problem we wish top solve is to learn a new policy $\pi^\star_{\Delta_{new}}$ such that

$$\pi^\star_{\Delta_{new}} = \arg\max_{\pi} \eta_{\Delta_{new}}(\pi). \tag{3}$$

3 Solution Approach

3.1 Translation of the Replicator Equation

Representation of populations and the fitness equivalent: We represent the probability distribution over actions in a given state as a population, where each type of population corresponds to a possible action. For a system comprising n states with m possible actions in each state, we effectively have n distinct populations of m types each. This results in m^n different types of individuals in the aggregated state population.

The fitness function measures the reproductive success of strategies based on payoffs from interactions, similar to utility in classical game theory, or how in RL, the expected return measures the long-term benefits of actions based on received rewards. Both serve as optimization criteria: strategies or policies are chosen to maximize these cumulative success measures to guide them towards optimal behavior. Therefore, in our model, the fitness for a state $f(s)$ corresponds to the expected return from that state, equivalent to the value function $v(s)$, and the fitness for a state-action pair $f(s,a)$ corresponds to the expected return from taking action a in state s, equivalent to the action-value function $q(s,a)$.

Under the assumption of sparse rewards—where significant rewards are received only upon reaching specific states or goals—$f(s)$ is defined as $\mathbb{E}[f(\tau_s)]$, the expected return across all trajectories through state s. Likewise, $f(s,a)$ is defined as $\mathbb{E}[f(\tau_{(s,a)})]$, the expected return across trajectories involving the state-action pair (s,a).

$$q(s,a) = f(s,a) \approx \mathbb{E}[f(\tau_{(s,a)})], \tag{4}$$

$$v(s) = f(s) \approx \mathbb{E}[f(\tau_{(s)})] \tag{5}$$

[1] Let $\beta = D_{TV}(\Delta||\Delta_{new})$, i.e. the total variation distance between the transition dynamics of the old and new environments; then β is bounded as $\beta \leq \beta_{hi}$. For our experiments, $\beta_{hi} = 0.4$.

Policy Update Mechanism: The replicator equation can be adapted to update the probability of selecting certain actions based on their relative performance compared to the average. The adaptation of the replicator equation is as follows:

$$\pi^{i+1}(s,a) = \frac{\pi^i(s,a)f(s,a)}{\sum_{a'\in A}\pi^i(s,a')f(s,a')} \tag{6}$$

where $\pi^i(s,a)$ is the probability of action a in state s, in the i^{ith} iteration, and π^{i+1} represents the policy int he $i+1^{th}$ iteration.

Lemma 1. *Policy update in Eq. (6) encodes the replicator dynamics in Eq. (2).*

The proof of the above lemma follows from the observation that for a state s, and a specific action a_i, the replicator equation 2 in our setting would look like[2]:

$$x_{s,a_j}(i+1) = x_{s,a_j}(i)\frac{f(s,a_j)}{\sum_{a\in A} f(s,a)\cdot x_{(s,a)}(i)} \tag{7}$$

By our representation of policies as populations, we see that $\pi^i(s,a_j)$ is equivalent to $x_{s,a_j}(i)$, and so Eq. (6) follows from Eq. (7). The policy update in Eq. (6) is just a simultaneous application of parallel replicator equations to all states (being updated in a given iteration).

This rule essentially captures the essence of the replicator dynamic by adjusting the probability of action a in state s proportionally to its performance relative to the average performance of all actions in that state. The normalizing factor in the denominator ensures that the updated policy remains a valid probability distribution, aligning with the principle of the replicator dynamic where strategy frequencies within a population must sum to one.[3]

Theorem 2. *The algorithm (ERPO) that employs the policy update specified in Eq. (6), ensures that the value of each state monotonically improves with each iteration, converging to an optimal policy under assumptions of sparse rewards.*

Proof. We note that our algorithm employs a batched Monte Carlo-style sampling approach, collecting multiple trajectories in each batch. We assume that each batch is sufficiently large to ensure that the estimated values of the v and q functions closely approximate their true values, so that Eq. (5) and Eq. (4) hold. We also assume that each state is visited at least once in each batch.

We define the action-value function and value function under policy π^i, accounting for transition probabilities:

$$q^i(s,a) = \sum_{s'\in S}\Delta(s,a,s')\left(r(s,a,s') + \gamma v^i(s')\right),$$

$$v^i(s) = \sum_{a\in A}\pi^i(s,a)q^i(s,a). \tag{8}$$

We now partition the set of actions A into A_h and A_l such that:

$$A_h = \{a_h \in A \mid q^i(s,a_h) \geq v^i(s)\}$$

[2] Assuming the fitness function is not time-dependent

[3] The original replicator equation describes the evolution of strategy proportions within a population, not the absolute numbers of individuals employing each strategy. This focus on proportions makes it a suitable model for normalizing factors in our policy update equation. Therefore, the replicator dynamic's mechanism, which adjusts strategy frequencies based on relative fitness, is analogous to adjusting the probability of action selections in relation to their expected return, thus ensuring that $\pi(s,a)$ remains a normalized probability distribution.

$$A_l = \{a_h \in A \mid q^i(s, a_l) < v^i(s)\}.$$

Splitting into contributions from A_h and A_l:

$$v^i(s) = \Sigma_{a_h \in A_h} \pi^i(s, a_h) q^i(s, a_h) + \Sigma_{a_l \in A_l} \pi^i(s, a_l) q^i(s, a_l) \tag{9}$$

Similarly,

$$v^{i+1}(s) = \Sigma_{a_h \in A_h} \pi^{i+1}(s, a_h) q^i(s, a_h) + \Sigma_{a_l \in A_l} \pi^{i+1}(s, a_l) q^i(s, a_l) \tag{10}$$

By our sparse reward assumption from Eqs. (4), (5) and (8) we can state that:

$$v^i(s) = f(s) = \sum_{a' \in A} \pi^i(s, a') f(s, a') \tag{11}$$

And the policy update equation can now be modified as:

$$\pi^{i+1}(s, a) = \pi(s, a) \left[\frac{q^i(s, a)}{v^i(s)} \right] \tag{12}$$

By definition: $q^i(s, a_h) \geq v^i(s)$ and $q^i(s, a_l) < v^i(s)$, and from Eq. (12) we get:

$$\pi^{i+1}(s, a_h) \geq \pi^i(s, a_h) \ ; \pi^{i+1}(s, a_l) < \pi^i(s, a_l)$$

the updated policy increases the probability of selecting a_h and decreases the probability of selecting a_l, so from Eqs. (9) and (10) we get:

$$v^{i+1}(s) \geq v^i(s)$$

Therefore, we show that a policy iteration algorithm based on this update rule guarantees that each state's value monotonically improves, ensuring convergence to the optimal policy.[4] □

Remark. Our algorithm operates in a Markovian framework, meaning state transitions depend only on the current state and action, without influence from past states/actions. Consequently, the replication of strategies and policy/value improvements can be applied to all states independently, where each state-action pair is updated without interference from the updates of the others. This facilitates parallel improvements across all states towards an optimal policy.

3.2 *Evolutionary Robust Policy Optimization (ERPO)*

Our algorithm uses batch-based updates: We initialize our training policy to be a weighted combination of the old optimal policy $\pi^\star$, and our new policy π_{new} - which is initially random. We sample trajectories as part of a batch, and in doing so we make sampling assumptions as mentioned earlier, and the state-action pairs in these trajectories are updated according to the update rule. The return for the $i + 1^{th}$ iteration is set as the return under the training policy, and the training policy is updated, to take into account the update to π_{new}. The weight assigned to the old policy is decremented with each iteration. This process is repeated until our termination condition $(\eta^{i+1} - \eta^i > \delta)$ is met. The termination condition checks if the expected return across all states is changing over our batch runs, and when the difference in the expected returns across consecutive batches is minimal, we say that the algorithm has converged.

[4] Convergence to an optimal policy in this setting with an evolutionary update is analogous to the concept of convergence to an evolutionarily stable strategy (ESS). An ESS is a strategy that, if adopted by a population, cannot be invaded by any alternative strategy that is initially rare. This implies that in this setting, once an optimal policy is reached, it cannot be outperformed easily by any other policy (under standard ESS assumptions [2]) thereby ensuring that the agent's behavior is robust against most changes and variations in the strategy space.

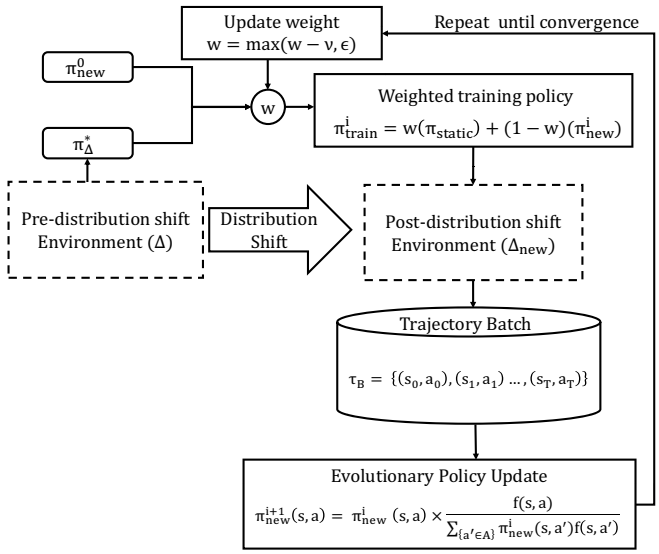


Figure 2: Outline of the ERPO methodology.

Algorithm 1 EVOLUTIONARY ROBUST POLICY OPTIMIZATION

Input:
Optimal policy $\pi^\star_\Delta = \arg\max_\pi \eta_\Delta(\pi)$
Initialize $\forall s \in S, a \in A : \pi^0_{new}(s, a) = \frac{1}{|A|}$
Hyperparameters $\epsilon, \nu \in (0, 1), \delta > 0$
Output: Optimized policy $\pi^\star_{new}$
Initialize $i \leftarrow 0$, η^0, $\pi^0_{train} \leftarrow w\pi^\star_\Delta + (1 - w)\pi^{i+1}_{new}$
repeat
  for each episode in batch b = 1 to B **do**
    Generate trajectory $\tau_b \sim \pi_{train}$
    Append trajectory τ_b to batch
  for all trajectories τ_b in batch **do**
    for each $(s, a) \in \tau_b$ **do**
      Update $\pi^{i+1}_{new}(s, a) = \frac{\pi^i_{new}(s,a) f(s,a)}{\sum_{a' \in A} \pi^i_{new}(s,a') f(s,a')}$
  Update expected return: $\eta^{i+1} \leftarrow \eta_{\Delta_{new}}(\pi^i_{train})$
  Update training policy $\pi^{i+1}_{train} \leftarrow w\pi^\star_\Delta + (1 - w)\pi^{i+1}_{new}$
  Decrement $w \leftarrow max(w - \nu, \epsilon)$;
  Increment $i \leftarrow i + 1$
until Convergence criteria: $(\eta^{i+1} - \eta^i \leq \delta)$)
return $\pi*_{new} \leftarrow \pi_{train}$

4 Experiments

Benchmarks: In our empirical analysis, we benchmark our approach against a selection of established reinforcement learning algorithms. The comparison is conducted under two different scenarios: (1) each RL algorithm is allowed to train on the modified environment from scratch, (2) we obtain pre-trained corresponding to the optimal policy, and then train them over the modified environment[5]. Each baseline scenario is described in detail below:

- **PPO** [26]: Standard Proximal Policy Optimization, re-trained from scratch in the new environment.
- **DQN** [15]: Deep Q-Network, also re-trained from scratch in the new environment.
- **A2C** [16]: Advantage Actor-Critic, re-trained from scratch in the new environment.

[5] We use the term 'model' to describe the specific neural network(s) used in each RL algorithm; for example, for PPO, this means the policy network.

- **PPO-B, DQN-B, A2C-B**: These baselines correspond to models that are trained by warm-starting the training in the new environment with the old optimal policy.
- **PPO-DR** [21]: Proximal Policy Optimization with Domain Randomization, enhances robustness by training across varied environments.

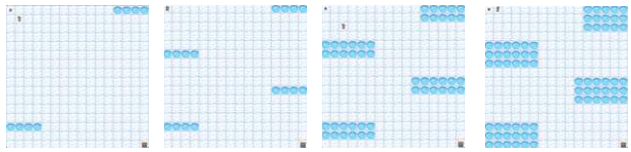
Implementations for these algorithms were sourced from the 'stable-baselines3' library [23], with hyperparameter optimization facilitated by the 'optuna' library [1]. All experiments were conducted on a high-performance computing cluster[6].

Environments: All the benchmarks are tested on the *FrozenLake*, *CliffWalking*, and *Taxi* environments in Open AI gymnasium, *Minigrid's Distribution Shift* environment [5] and a version of the *Minigrid: Empty* environment customized with walls and lava. We remark that we use larger and more complex versions of the standard environments to test our algorithm properly. More precisely we vary the Total Variation Distance between Δ and Δ_{new} i.e. the transition dynamics of the old and new environments between 0.15 and 0.4.

1. *FrozenLake (**FL**):* The agent tries to navigate across a Frozen Lake to reach a goal. The episode terminates when the agent enters a hole and drowns, or reaches the goal. We have a base model with few holes and three additional levels with increasing grid environment occupied by holes, indicated with the darker blue in Fig. 3.
2. *CliffWalking (**CW**)*: The agent starts on the bottom left and must reach the goal location (bottom right) while avoiding 'cliff' locations (indicated in brown), otherwise it is returned to the start position. We have a base model with one row of cliffs (similar to the standard model), and three additional levels with increasing cliff area (Fig. 4).
3. *Taxi (**TX**)*: The agent (a taxi) must pick up and drop a passenger from designated stations (indicated in boxes of red, green, blue and yellow). Dividers prevent the taxi from turning left or right, forcing it to take a more circuitous route and make U-turns. The base model has no dividers, and additional levels have increasing numbers of dividers (see Fig. 7).
4. *Minigrid: DistributionShift (**MGDS**)*: The purpose is to test the ability to generalize across two variations of the environment. The episode terminates when the agent reaches the goal or lava. We have three levels of environments with increasing grid areas occupied by lava. See Fig. 5.
5. *Minigrid: Walls&Lava (**MGWL**)*: Lastly, we test the ability to navigate in the presence of walls that block the agent's vision and movement, or lava that terminates the episode, or both. The base model is an empty grid, Level 1 has lava, Level 2 has walls, and Level 3 has both. See Fig. 6.

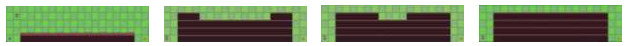
Implementation Details: We assume that the optimal policy in the original environment is obtained using PPO (this can be changed to other algorithm). The base PPO model is allowed learns over a minimum of 10^4 and a maximum of 10^6 timesteps – the number of steps required to converge to (close to) π^*_Δ varies across environments. As we induce distribution shifts, we need to pick reward functions that are sensible across all instantiations of any environment. The reward functions for each environment are standard across all the models of all the algorithms used. Further details of the reward functions are presented in the results, and other details such as the hyperparameters can be found in the supplementary material. [7]

Remark. We note that the reward function modification adheres to the principles outlined in [19], utilizing transformations of the reward function that maintain policy invariance. Specifically, the rewards are structured as potential-based transformations, where the modified reward function is given by $r'(s,a,s') = r(s,a,s') + \gamma\Phi(s') - \Phi(s)$, with Φ being a potential function. Despite incorporating scaled rewards, these transformations preserve the optimality of the policies as the potential-based adjustments ensure the fundamental characteristics of the original reward system are maintained. This confirms the robust application of our model and validates its efficacy across varied and complex reward structures.



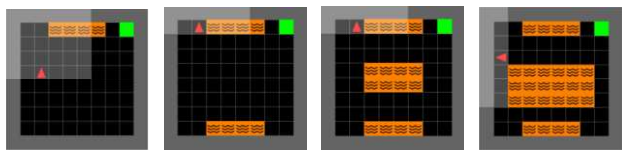
(a) FL: Base (b) FL: L1 (c) FL: L2 (d) FL: L3

Figure 3: Frozen Lake Environments



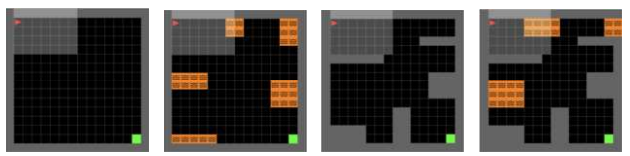
(a) CW: Base (b) CW: L1 (c) CW: L2 (d) CW: L3

Figure 4: Cliff-Walking Environments



(a) MGDS: B (b) MGDS: L1 (c) MGDS: L2 (d) MGDS: L3

Figure 5: Minigrid:DistributionShift Environments



(a) MGWL: B (b) MGWL: L1 (c) MGWL: L2 (d) MGWL: L3

Figure 6: Minigrid:Walls&Lava Environments

(a) TX: B (b) TX: L1 (c) TX: L2 (d) TX: L3

Figure 7: Taxi Environments

5 Results

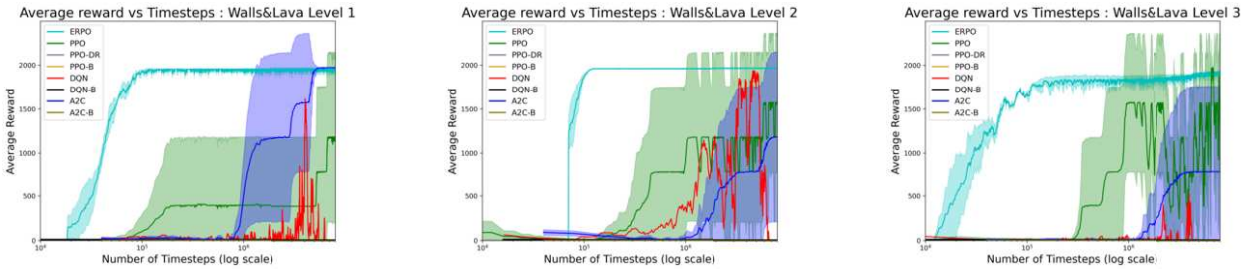
We present the results of each environment for ERPO and the other baseline algorithms. We note that the performance of ERPO does not vary much with increasing levels of difficulty, even when the new environment is drastically different and much more difficult to navigate than the base environment, while the other algorithms suffer.

Comparison with models trained from scratch and Domain Randomization: ERPO significantly outperforms the other algorithms in terms of timesteps required for convergence. PPO-DR and A2C are the closest competitors, yet they still require up to an order of magnitude more timesteps than ERPO in the Walls&Lava environment. [8] The results from the Taxi environment indicate that PPO-DR has

[6] The computational resources included nodes with dual 8-16 core processors, 16 CPUs, and 32GB of memory per node.

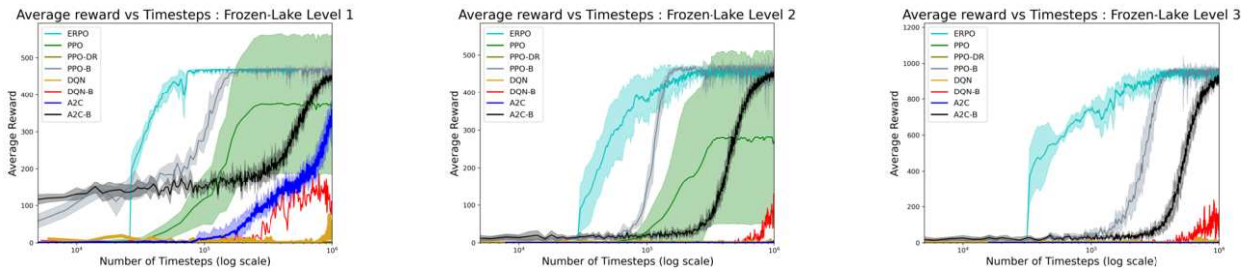
[7] https://github.com/sherylpaul/ERPO

[8] A2C shows results later than the other algorithms in the Walls&Lava environment because of large batch size over 8 environments, so results are indicated only after batch_size $\times$ 8 timesteps.



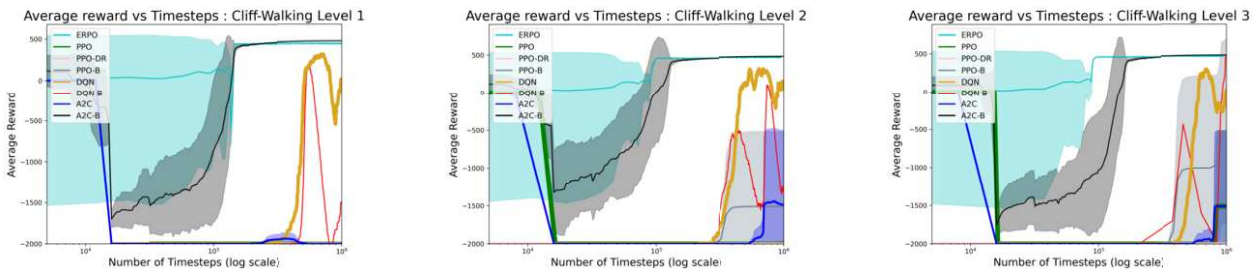
(a) Level 1 (b) Level 2 (c) Level 3

Figure 8: Walls&Lava Environments: Subfigures (a), (b), and (c) indicate the results for levels 1, 2 and 3 (see Fig. 6). The base figure is the original environment, and levels 1 to 3 indicate versions of the environment with certain features altered that induce progressively increasing distribution shift. The agent has to navigate a gridworld of increasing complexity with lava, walls or both. It has 2000 timesteps to reach the goal ($r = 0$ for each step) and gets reward $r = 2000 - t$ upon reaching the goal (where t is the current timestep). ERPO surpasses other models, with steadied performance even as levels increase.



(a) Level 1 (b) Level 2 (c) Level 3

Figure 9: Frozen-Lake Environments: (See Fig. 3) The agent (elf) is given 500 steps to reach the goal ($r = 0$ for each step) and given a reward $r = 500 - t$ upon reaching the goal (where t is the current timestep). ERPO consistently maintains a strong performance, particularly at higher levels. PPO-DR and A2-C B demonstrate robustness.



(a) Level 1 (b) Level 2 (c) Level 3

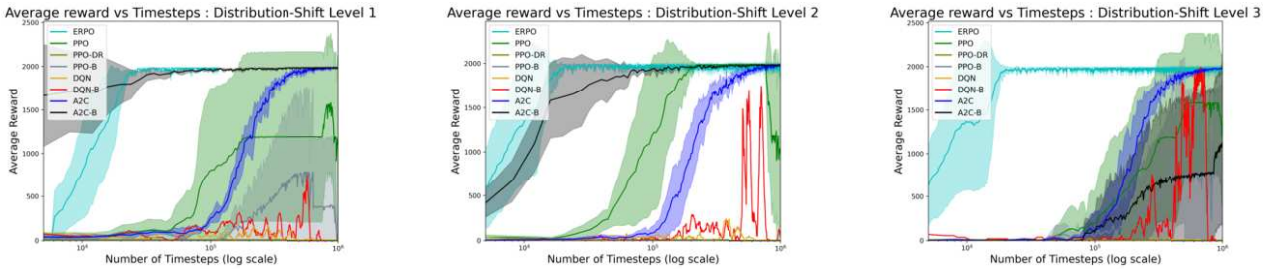
Figure 10: Cliff-Walking Environments: (See Fig. 4) The agent is allowed 2000 steps to get from the start to the goal ($r = -1$ for each step) and given a reward $r = +500$ for reaching the goal. ERPO outperforms other methods, at higher levels. Notably, baselines trained over pre-learned models tend to fluctuate, indicating challenges in adapting to new environments. A2C-B comes close and outperforms ERPO in Level 1.

the closest performance to ERPO but still takes longer to converge. In the modified CliffWalking environments, most algorithms struggle to converge, highlighting the increased difficulty due to the larger cliff area and being returned to the starting position for entering it. The FrozenLake environment presents a more favorable scenario for A2C and PPO, but they worsen significantly as the levels increase, empirically demonstrating that ERPO adapts better.

Comparison with models trained over pre-trained (base) models: In the distribution-shifted environment, the models trained over baselines perform slightly better than their counterparts trained from scratch. In the Taxi environment, PPO-B shosw relatively close performance to ERPO. The large cliff area and long episodes in the CliffWalking environment prove challenging, with most algorithms failing to converge. However, A2C-B performs nearly as well as ERPO, though it takes slightly longer to achieve convergence. The FrozenLake environment shows that PPO-B and A2C-B perform competently.

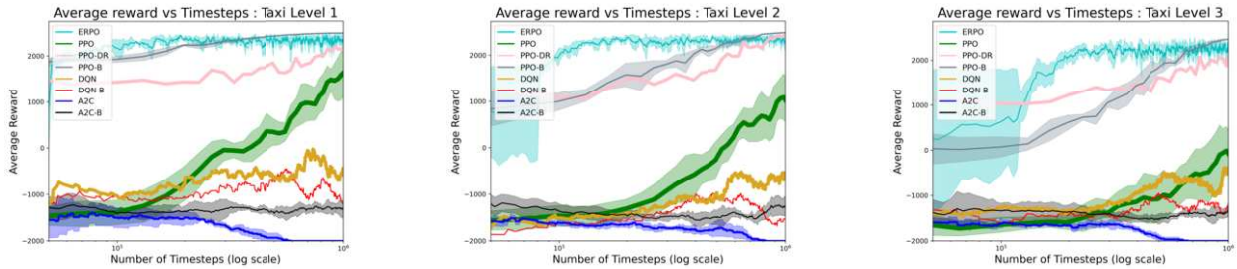
Benchmarking against Heuristic Search Methods: In addition to comparing learning algorithms, we benchmark baseline heuristic search algorithms, specifically A^* and IDA^*, in the FrozenLake and CliffWalking environments. These comparisons use Manhattan distance as the heuristic, with other costs aligned to the previously described reward structures. It is important to note that these comparisons serve as baselines, not direct competitors, as the information available to these algorithms differs, making a like-for-like comparison unfair.

In the FrozenLake environment, A^* performs poorly due to its inability to account for proximity to holes, resulting in significant penalties. The cost incurred by A^* ranges from -$5K$ to -$15K$ across different versions of FrozenLake. Conversely, IDA^* performs well on Level 1 with a reward of approximately $2K$, but declines to -$5K$ on Level 3.



(a) Level 1 (b) Level 2 (c) Level 3

Figure 11: Distribution-Shift Environments: (See Fig. 5) The agent has to navigate a gridworld of increasing complexity with lava. It has 2000 timesteps to reach the goal ($r = 0$ for each step) and $r = 2000 - t$ upon reaching the goal (where t is the current timestep). ERPO outperforms other methods while A2C-B comes close. Besides that PPO and A2C perform well too. The increased environmental complexity from Level 1 to Level 3 is evident, with all models facing greater challenges as the level increases.



(a) Level 1 (b) Level 2 (c) Level 3

Figure 12: Taxi Environments: (See Fig. 7) The agent (taxi) is allowed to take 2000 steps in an episode ($r = -1$ for each step), and gets a reward $r = +2500$ for correct pick up and drop. ERPO outperforms other algorithms with PPO-B and PPO-DR models showing good results.

In the CliffWalking environment, IDA^* consistently achieves rewards of approximately $2K$ across all levels, though this success requires the starting node to be positioned near the cliff's edge. In contrast, A^* struggles, with rewards ranging from -200 to -300.

While vanilla heuristic search methods may underperform in stochastic environments, methods like stochastic A^* might be more suitable. Additionally, extending these methods to continuous spaces via function approximators is challenging and requires an admissible heuristic. A key difference is that ERPO and other learning-based methods generate policies that can generalize to any start location within the grid, whereas heuristic-based approaches may need to restart the search when encountering previously unexplored states.

Preliminary Results on custom Environments: In addition to the results on standard gym environments, we also conducted experiments in custom gym environments with a similar reach-avoid mission. The agents in these environments are assigned randomly to one of many pre-determined start locations, and must reach one of the goal locations whilst avoiding obstacles. One sample result for a 100x100 grid world with 20% new obstacles is as follows: We leave the extension to larger custom grids with varying levels of obstacle density for future work.

Algorithm	A*	Q-learning	PPO	ERPO
Path Length	118.41	118.00	122.43	116.05

Table 1: Comparison of Algorithm Performance

Analysis: Our observations indicate a distinct advantage of ERPO over traditional reinforcement learning algorithms even when trained over the pre-trained models, and domain randomization methods. While PPO, PPO-DR, and A2C utilize batch-wise updates, and DQN depends on episode-wise updates, these algorithms generally treat each step within a batch or episode as equally significant for the purpose of policy updates. This approach can dilute the impact of particularly successful or unsuccessful trajectories on the overall learning process. In contrast, ERPO places emphasis on trajectories that significantly deviate from the norm — either by outperforming or underperforming compared to the rest of the batch and prioritizes learning from those that are the most informative. This selective update mechanism ensures that ERPO rapidly identifies and leverages the most effective strategies. As a result of this approach, the fittest trajectories become increasingly predominant in the batch over just a few training episodes. Thus, ERPO leads to a faster and more efficient convergence towards optimal policies.

6 Discussion

Limitations and Future Work: Our set up is limited to discrete state-action spaces. We are working on an extension that works with continuous spaces. This will be carried out with function approximation using radial basis functions that also update the policies of states within a certain distance of the state we are updating. Additionally, because we normalize the probability distribution across actions of a given state, a continuous model would work instead along with a probability density function that can be updated using Dirac delta functions. Our set up is also limited to single agent models (unless extended with independent learning). We are working on extensions that can combine other game-theoretic solution concepts for cooperative multi-agent learning.

Conclusion: This paper presents a new approach to incrementally adapt the optimal policy of an autonomous agent in an environment that experiences large distirbution shifts in the environment dynamics. Our algorithm uses principles from evolutionary game theory (EGT) to adapt the policy and our policy update can be viewed as a version of replicator dynamics used in EGT. We provide theoretical convergence guarantees for our algorithm and empirically demonstrate that it outperforms several popular RL algorithms, both when the algorithms are warm-started with the old optimal policy, and when they are re-trained from scratch.

References

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019.

[2] J. M. Alexander. Evolutionary game theory. The Stanford Encyclopedia of Philosophy (Summer 2021 Edition), 2021. URL https://plato.stanford.edu/archives/sum2021/entries/game-evolutionary/.

[3] M. Bellusci, N. Basilico, and F. Amigoni. Multi-agent path finding in configurable environments. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 159–167, 2020.

[4] D. Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific, 2019.

[5] M. Chevalier-Boisvert, B. Dai, M. Towers, R. de Lazcano, L. Willems, S. Lahlou, S. Pal, P. S. Castro, and J. Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *CoRR*, abs/2306.13831, 2023.

[6] E. Derman, D. Mankowitz, T. Mann, and S. Mannor. A bayesian approach to robust reinforcement learning. In *Uncertainty in Artificial Intelligence*, pages 648–658. PMLR, 2020.

[7] Y. Elmaliach, N. Agmon, and G. A. Kaminka. Multi-robot area patrol under frequency constraints. *Annals of Mathematics and Artificial Intelligence*, 57(3):293–320, 2009.

[8] S. Guo, X. Zhang, Y. Zheng, and Y. Du. An autonomous path planning model for unmanned ships based on deep reinforcement learning. *Sensors*, 20(2):426, 2020.

[9] T. A. Han. Emergent behaviours in multi-agent systems with evolutionary game theory. *AI Communications*, 35(4):327–337, 2022.

[10] Y. Jiang, C. Li, W. Dai, J. Zou, and H. Xiong. Monotonic robust policy optimization with model discrepancy. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 4951–4960. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/jiang21c.html.

[11] A. Kadian, J. Truong, A. Gokaslan, A. Clegg, E. Wijmans, S. Lee, M. Savva, S. Chernova, and D. Batra. Sim2real predictivity: Does evaluation in simulation predict real-world performance? *IEEE Robotics and Automation Letters*, 5:6670–6677, 2020.

[12] J. Li, Z. Chen, Y. Zheng, S.-H. Chan, D. Harabor, P. J. Stuckey, H. Ma, and S. Koenig. Scalable rail planning and replanning: Winning the 2020 flatland challenge. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 477–485, 2021.

[13] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. S. Kumar, and S. Koenig. Lifelong multi-agent path finding in large-scale warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11272–11281, 2021.

[14] M. A. Luna, M. S. Ale Isaac, A. R. Ragab, P. Campoy, P. Flores Peña, and M. Molina. Fast multi-uav path planning for optimal area coverage in aerial sensing applications. *Sensors*, 22(6):2297, 2022.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning, 2013.

[16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[17] R. Morris, C. S. Pasareanu, K. Luckow, W. Malik, H. Ma, T. S. Kumar, and S. Koenig. Planning, scheduling and monitoring for airport surface operations. In *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[18] A. Mukhopadhyay and S. Chakraborty. Replicator equations induced by microscopic processes in nonoverlapping population playing bimatrix games. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31 (2), Feb. 2021. ISSN 1089-7682. doi: 10.1063/5.0032311. URL http://dx.doi.org/10.1063/5.0032311.

[19] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287. Citeseer, 1999.

[20] S. Paul and J. V. Deshmukh. Multi agent path finding using evolutionary game theory, 2022. URL https://arxiv.org/abs/2212.02010.

[21] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2018. doi: 10.1109/icra.2018.8460528. URL https://doi.org/10.1109%2Ficra.2018.8460528.

[22] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta. Robust adversarial reinforcement learning, 2017.

[23] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[24] A. Rajeswaran, S. Ghotra, S. Levine, and B. Ravindran. Epopt: Learning robust neural network policies using model ensembles. *CoRR*, abs/1610.01283, 2016. URL http://arxiv.org/abs/1610.01283.

[25] W. H. Sandholm. *Evolutionary Game Theory*, pages 3176–3205. Springer New York, New York, NY, 2009. ISBN 978-0-387-30440-3. doi: 10.1007/978-0-387-30440-3_188. URL https://doi.org/10.1007/978-0-387-30440-3_188.

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

[27] E. Smirnova, E. Dohmatob, and J. Mary. Distributionally robust reinforcement learning, 2019.

[28] J. Smith. *Evolution and the Theory of Games*. Cambridge University Press, 1982. ISBN 9780521288842. URL https://books.google.com/books?id=Nag2IhmPS3gC.

[29] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[30] C. Tessler, Y. Efroni, and S. Mannor. Action robust reinforcement learning and applications in continuous control. In *International Conference on Machine Learning*, pages 6215–6224. PMLR, 2019.

[31] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis. Gymnasium, Mar. 2023. URL https://zenodo.org/record/8127025.

[32] K. Tuyls and S. Parsons. What evolutionary game theory tells us about multiagent learning. *Artificial Intelligence*, 171(7):406–416, 2007.

[33] S. Varambally, J. Li, and S. Koenig. Which mapf model works best for automated warehousing? In *Proceedings of the International Symposium on Combinatorial Search*, volume 15, pages 190–198, 2022.

[34] H. Zhang, H. Chen, D. Boning, and C.-J. Hsieh. Robust reinforcement learning on state observations with learned optimal adversary, 2021.

[35] X. Zhou, P. Wu, H. Zhang, W. Guo, and Y. Liu. Learn to navigate: Cooperative path planning for unmanned surface vehicles using deep reinforcement learning. *IEEE Access*, 7:165262–165278, 2019. doi: 10.1109/ACCESS.2019.2953326.

[36] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou. Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(11):13344–13362, 2023. doi: 10.1109/TPAMI.2023.3292075.