REvolveR: Continuous Evolutionary Models for Robot-to-robot Policy Transfer

Xingyu Liu [1] **Deepak Pathak** [1] **Kris M. Kitani** [1]

Abstract

A popular paradigm in robotic learning is to train a policy from scratch for every new robot. This is not only inefficient but also often impractical for complex robots. In this work, we consider the problem of transferring a policy across two different robots with significantly different parameters such as kinematics and morphology. Existing approaches that train a new policy by matching the action or state transition distribution, including imitation learning methods, fail due to optimal action and/or state distribution being mismatched in different robots. In this paper, we propose a novel method named *REvolveR* of using continuous evolutionary models for robotic policy transfer implemented in a physics simulator. We interpolate between the source robot and the target robot by finding a continuous evolutionary change of robot parameters. An expert policy on the source robot is transferred through training on a sequence of intermediate robots that gradually evolve into the target robot. Experiments on a physics simulator show that the proposed continuous evolutionary model can effectively transfer the policy across robots and achieve superior sample efficiency on new robots. The proposed method is especially advantageous in sparse reward settings where exploration can be significantly reduced. Code is released at https://github.com/xingyul/revolver.

1. Introduction

A popular paradigm in learning robotic skills is to leverage reinforcement learning (RL) algorithms to train a policy for every new robot in every new environment from scratch. This is not only inefficient in terms of sample efficiency but also often impractical for complex robots due to an extremely large exploration space. How can one transfer a well-trained policy on one robot to another robot?

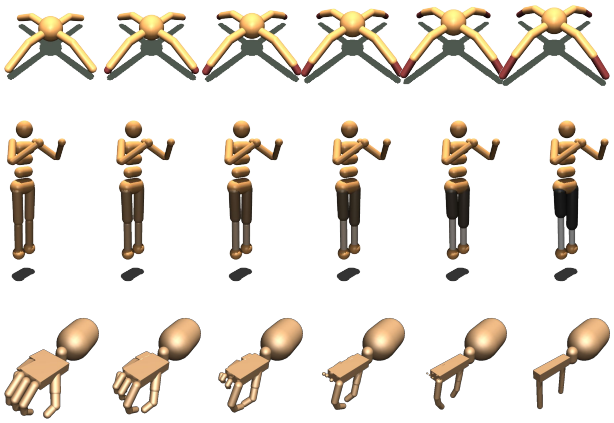


Figure 1. **Continuous robot evolution model** allows policy to be transferred from one robot to another robot. Upper row: an Ant robot continuously grow additional legs from the tip of its feet. Middle row: a Humanoid robot continuously changes the length and mass of its legs. Lower row: a dexterous gripper continuously shrink three of its fingers to evolve to a two-finger gripper. We show the robots at evolution parameters of 0.0, 0.2, 0.4, 0.6, 0.8 and 1.0 respectively from left to right in each row.

Past endeavors have explored two main directions for transferring policy between robots. Statistic matching Imitation learning (IL) methods train a new policy on the target robot with the aim of matching the behavior of the policy on a source robot. Methods that optimize to match the distribution of actions (Ross et al., 2011), state rollouts (Liu et al., 2019; Radosavovic et al., 2020), or reward function (Ng et al., 2000; Ho & Ermon, 2016) have been successful on robotic learning tasks on robot with **similar** dynamics. However, these methods are unable to deal with cases with **very large difference** in robot parameters and dynamics, since when mapped to the same state and action space, the robots could have very different optimal distributions of states or actions. An alternative to imitation learning is to learn the robot hardware dynamics together with the policy by encoding the robot hardware specifics with neural networks (Chen et al., 2018; Huang et al., 2020). However, to train such hardware-aware policies, it usually requires training diverse tasks on a huge number of robots in advance, which could be computationally prohibitive.

[1]The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA. Correspondence to: Xingyu Liu <xingyul3@cs.cmu.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

In this paper, we propose a new paradigm for policy transfer between robots. Our framework, named *REvolveR*, is based on defining a continuous evolution of robots, where both the robot morphology and kinematics are continually adjusted to allow transforming one robot (source) to another robot (target), as illustrated in Figure 1.

Specifically, the continuous evolutionary model interpolates two different robots by producing an infinite number of intermediate robots whose parameters are represented in continuous space. These intermediate robots act as the "bridge" for transferring the policy from the source robot to the target robot. We are able to evaluate any robot along this continuum using physics simulation. Then the policy is progressively trained on a sequence of intermediate robots whose robot parameters gradually evolve towards the target robot. Since the change of the evolved robot parameters and hardware dynamics is small enough, it is typically easy for the policy to adapt to the new robots. By the joint gradual evolution of robot hardware dynamics and the policy, we decompose the difficult robot-to-robot policy transfer problem into a sequence of policy fine-tuning problems that are much easier to solve.

Additionally, we propose several approaches that improve sample efficiency and stabilize training during the robot-to-robot policy transfer. To stabilize training, we propose a local randomized evolution strategy where in each training epoch, we randomly sample a set of robots over a small continuous range of robot agents. Over time, the set of robots gradually transform into the target robot. This allows the policy to adapt to a diverse set of robot transition dynamics within a local range. To improve sample efficiency, we propose an evolution reward shaping technique where we enforce larger weights on the reward received from more "evolved" robots to encourage the policy to adapt towards the target robots. We present theoretical results to show that this strategy improves the adaptation.

We develop the continuous robot evolution models on a diverse set of robots and demonstrate the effectiveness of the proposed policy transfer approach with three different RL algorithms. We showcase our REvolveR on three MuJoCo Gym environments (Brockman et al., 2016) with dense reward. Our method achieves significantly higher performance than direct policy transfer and imitation learning baselines. We also experiment on Hand Manipulation Suite tasks (Rajeswaran et al., 2018) in sparse rewards setting. While methods for learning from human demonstration completely fails, our method can still transfer the policy in the challenging sparse reward setting.

We expect the new problem of robot-to-robot policy transfer as well as the proposed REvolveR framework to be the new paradigm for inter-robot transfer learning and inspire research in related domains.

2. Related Work

Morphological Evolution Ideas centered around evolutionary mechanisms to develop complex robot morphologies dates back to the work from Von Neumman (Von Neumann et al., 1966). The series of seminal work from Karl Sims showed how genetic algorithms can be leveraged to develop both complex morphologies as well as their controllers through an evolutionary optimization process (Sims, 1994a;b). Morphological changes at evolutionary scales have also been related to development during the life of the organism and how are these related to each other (Clune et al., 2012; Kriegman et al., 2018). Our work instead assumes that the source and target robots are given and figures out how to evolves latter from the former to transfer the controller policy.

Learning Controllers for Diverse Robot Morphology It is often difficult to design controllers for complex robots. Learning controllers via a curriculum of robots with gradually growing complexity provides a path towards controlling high-dimensional robot morphologies. This concept has been used by a recent line of work that grows control and morphology simultaneously. For instance, Pathak et al. (2019) learns to control and develop different morphologies simultaneously to build agents that can generalize to new scenarios using dynamic graph neural networks. Vanilla GNNs (Scarselli et al., 2009) have been used to control diverse robot morphologies in NerveNet (Wang et al., 2018) to control different robots obtained by growing the limbs within topology (Wang et al., 2019b; Hejna III et al., 2021) or across topology (Gupta et al., 2021). Learning-driven evolution could be used to improve the design as well of the agent (Cheney et al., 2014; Ha et al., 2017; Ha, 2018; Schaff et al., 2018; Pan et al., 2021). Similarly, one could also evolve the environment itself too (Wang et al., 2019a). Another rich approach to improve the design is to evolve the robot with a predefined grammar of physical components (Zhao et al., 2020). In contrast to these works, we do not co-develop the controller with morphology but transfer the policy from a source robot to target robot by simulating an evolutionary process. Our approach can be applied to any given robot without being limited to the robots that appear as a biproduct of co-evolution.

Closer to ours is the line of work that tries to build controllers that can work across large kind of robots. Huang et al. (2020) leverages modularity using graph neural networks across limbs of robots to train agent-agnostic policies, which have later been replaced by transformer architectures (Kurin et al., 2020). Another simple way is to condition on the hardware one-hot vector if topology remains the same (Chen et al., 2018). Hierarchical controllers have also been shown to be effective while transferring across morphologies (Hejna et al., 2020). Our work differ from these

prior works in the sense that we assume that we are already given a good controller for some morphology and we use that to generate a controller for some new robot rather than training from scratch.

Modular Robotics Another closesly related area in robotics is that of building modular components which can be used to build diverse robot morphologies. These modular systems can either be self-configurable (Stoy et al., 2010; Murata & Kurokawa, 2007) or docked manually to build complex robotic shapes (Yim et al., 2000; Wright et al., 2007; Romanishin et al., 2013; Gilpin et al., 2008; Daudelin et al., 2018). Recent work in this direction uses model-based learning to build and control these modular robots (Whitman et al., 2021; 2020).

Our work converts discrete optimization to continuous optimization. Similar ideas can be found in differentiable neural architecture search (Zoph & Le, 2016; Liu et al., 2018) where neural architecture is equivalent to our robot architecture. Furthermore, our work can be viewed as a domain transfer between two MDP domains.

3. Preliminary and Problem Statement

MDP Preliminary We consider an infinite-horizon Markov Decision Process (MDP) defined by $M = (\mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is transition dynamics with $\mathcal{T}(s, a, s')$ the probability of transitioning from state s to s' when action $a \in \mathcal{A}$ is taken, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward associated with taking action a at state s, and γ is the discount factor. The set of all MDPs is $\mathcal{M}$. We assume both the state space $\mathcal{S}$ and the action space $\mathcal{A}$ are continuous.

A policy π is a function that maps states to a probability distribution over actions where $\pi(a \mid s)$ is the probability of taking action a at state s. Given a MDP M with transition $\mathcal{T}$ and policy π, let $V^{\pi,M}$ be the value function on the model M and policy π, defined as:

$$V^{\pi,M}(s) = \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim M(\cdot|s_t, a_t)}} [\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s] \quad (1)$$

The optimal policy π^*_M is the policy that maximize the value function on MDP M, defined as $\pi^*_M(s) = \arg\max_\pi V^{\pi,M}(s)$. The objective of MDP optimization is to find the optimal policy under a given MDP.

Problem Statement We consider the problem of transferring a *source* policy trained for one robot to a new *target policy* that must work on a different robot. To limit the scope of this problem, we make the assumption that the two robots share the same state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $\mathcal{R}$ and discount factor γ. The main difference between the source policy and target policy is that they are optimal for different transition dynamics.

Formally, we consider two robots represented by two MDP M_S (*source*) and M_T (*target*) respectively. We assume the state and action space of M_S and M_T are shared. Given a well-trained expert policy π_{M_S} on a source robot M_S, the goal is to find the optimal policy $\pi^*_{M_\text{T}}$ on a target robot M_T. Though an ordinary reinforcement learning algorithm could be used to find $\pi^*_{M_\text{T}}$, we would like to investigate using the information in $\pi^*_{M_\text{S}}$ to improve the sample efficiency as well as the final performance of π_{M_T}.

4. Method

In general, transferring the policy of one robot (source) to a different robot (target) can be challenging, especially when there is a large mismatch in the dynamics of the two robots (*e.g.*, different number of joints or limbs, extreme difference in limb length). However, when the difference between the dynamics of two robots is sufficiently small, we also hypothesize that it may be easier to directly transfer the policy of the source robot to the target robot. If this hypothesis is true, it stands to reason that by defining a sequence of micro-evolutionary changes of the source robots into the new dynamics of the target robot, we should be able to transfer the policy of the source robot to the target robot through incremental policy updates over that sequence.

Motivated by this hypothesis, our strategy is to define an evolutionary sequence of dynamics models that connects the source dynamics to the target dynamics. Then we will incrementally optimize the source policy by interacting with each model in the sequence until the policy is able to act (near) optimally under the target dynamics. With multiple steps of robot change and training, the robot could eventually evolve to the target robot and transfer the policy. However, the maximum amount of changes that can preserve sufficient task completion rate and reward is unknown and could be arbitrarily small. An overlarge change to the robot could bring it to a "trap" where it never receive enough reward again and completely fail in transferring the policy.

Our solution is to develop a continuous evolution model from the source to the target robot. The continuous model allows arbitrarily small changes towards the target robot to be made and hence transfer the policy with a smoothly developed curriculum. The overall idea is in Algorithm 1.

4.1. Continuous Robot Model Evolution

Given the source robot M_S and target robot M_T, we define a continuous function $E : [0, 1] \to \mathcal{M}$ such that $E(0) = M_\text{S}$ and $E(1) = M_\text{T}$. The function E returns an interpolation between two MDPs. Since we assume the same state, action

and reward for the source and target robots, the function E essentially produces a newly interpolated transition dynamics model. For any evolution parameter $\alpha \in (0, 1)$, $E(\alpha)$ is an intermediate robot between M_{S} and M_{T}. In general, interpolating two different robots requires both the morphology matching and kinematics interpolation.

Morphology Matching The first step of robot interpolation is two match the morphology of the two robots. The body and joint connection of a robot can be described by a kinematic tree. This step essentially finds the topological matching of the kinematic trees of the two robots. By determining the root nodes of both kinematic trees and if necessary, creating the missing nodes and/or edges, we can always find an one-to-one correspondence of node and edges between the two robots. The procedure is illustrated in Figure 2. In practice, however, to minimize the gap between source and target robots, we choose root nodes so that the adding of new nodes is minimal. For example, a two-finger robot gripper could be mapped to a five-finger dexterous hand by attaching three zero sized fingers and joints to the palm node. Creating new nodes and edges usually changes the state space $\mathcal{S}$ and action space $\mathcal{A}$ with zero insertions in the state and action vectors so that the original MDP transition dynamics $\mathcal{T}$ is not changed.

Kinematic Interpolation Given the correspondence in bodies and joints, the source and target robots may still have mismatch in other kinematics parameters that affects the physical dynamics, such as size and inertial of the bodies, and motor and damping of the joints etc. Suppose the morphology of source and target robots is matched and the kinematic parameters of the source and target robots are mapped to the same space. Function $E(\alpha)$ can define an interpolated robot by interpolation between all pairs of kinematics parameters. Formally, suppose the kinematic parameters of source and target robots are θ_{S} and θ_{T} in the same space. The parameter of the interpolated robot $M_\alpha = E(\alpha)$ is

$$\theta(\alpha) = (1 - f(\alpha))\theta_{\text{S}} + f(\alpha)\theta_{\text{T}} \tag{2}$$

where $f : [0, 1] \to [0, 1]$ is a continuous function and $f(0) = 0$, $f(1) = 1$ to ensure $\theta(0) = \theta_{\text{S}}$ and $\theta(1) = \theta_{\text{T}}$ so that $M_0 = M_{\text{S}}$ and $M_1 = M_{\text{T}}$. In this paper, we choose to use a simple linear interpolation of $f(\alpha) = \alpha$, though a more sophisticated interpolation strategy can also be adopted.

The above morphology matching and kinematic interpolation steps can be easily implemented by editing the robots' URDF or MJCF files in physics simulation engines such as MuJoCo (Brockman et al., 2016) and pyBullet (Coumans & Bai, 2016). Note that evolution parameter α not only represents the evolution of robot hardware specifics described by a real scalar, but also can describe the continuous change of more complex hardware specifics, such as the progress of continuous mesh deformation if the shapes of the two corresponding robot bodies are different.

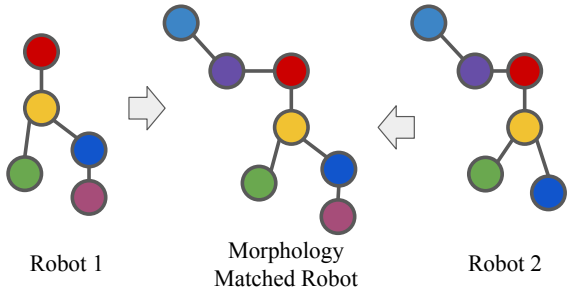


Figure 2. **Morphology matching of two robots.** Though the two robots may be different in morphology (i.e. kinematic tree topology), by properly choosing a root node (e.g. the yellow node), we can always add new nodes and edges to the kinematic tree of the robot(s) to match their morphology.

4.2. Policy Transfer on Continuously Evolving Robots

Suppose a well-trained policy $\pi_{E(0)}$ for source robot $E(0)$ is given. Instead of directly transferring $\pi_{E(0)}$ to robot $E(1)$, we decompose the problem into K phases of policy optimization. In k-th phase, the policy is trained on robot $E(\alpha_k)$ with evolution parameter $\alpha_k = \sum_{i=1}^{k} l_i$, where l_i is a small positive real number representing the progression of evolution parameter in i-th phase. The optimization objective in the $(k+1)$-th phase is

$$\pi_{E(\alpha_{k+1})} = \arg\max_{\pi} \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim M_{\alpha_{k+1}}(\cdot|s_t, a_t)}} \sum_t \gamma^t r_t \tag{3}$$

where $M_{\alpha_{k+1}} = E(\alpha_{k+1})$ is the next evolved robot and π is initialized to be $\pi_{E(\alpha_k)}$ at the start of the optimization. By definition of the problem, we have $\alpha_K = \sum_{i=1}^{K} l_i = 1$. The K and l_k values are set such that l_k is small enough so that during the policy optimization in Equation (3), there exists sufficient amount of positive reward in the rollouts to allow policy training. Ideally, l_k values are maximized so that the number of optimization phases K and the total number of RL iterations can minimized.

However, it is not possible to foresee the maximum allowed l_k before training the policy on robot $E(\sum_{i=1}^{k} l_i)$. In fact, there is a dilemma of trade-off between the value of l_k and the total number of RL iterations: with large l_k and aggressive progression of α_k, the policy may not receive enough positive reward from the new robot to be trained and adapted, and with small l_k and conservative progression of α_k, the policy may waste RL iterations on tiny robot changes.

Algorithm 1 Continuous Robot Evolution Policy Transfer

Notation Summary:
$\alpha \in [0, 1]$: robot evolution parameter
$E : [0, 1] \to \mathcal{M}$: continuous robot evolution model
$\pi_{E(0)}$: expert policy on the source robot $E(0)$
$\mathcal{R}$: replay buffer buffer
$\xi \in \mathbb{R}^+$: range of sampling of evolution parameters
$l_k \in \mathbb{R}^+$: progression of α in phase k, where $l_k < \xi$
$h \in \mathbb{R}^+$: evolution reward shaping factor

// initialize evolution parameter, policy and replay buffer
$\alpha \leftarrow 0, \pi \leftarrow \pi_{E(0)}, \mathcal{R} \leftarrow \emptyset$
while $\alpha < 1$ **do**
  for epoch **in** $0, 1, \ldots, N_e$ **do**
    // sample an intermediate robot
    $\beta \sim \text{Uniform}(\alpha, \min\{\alpha + \xi, 1\})$
    $M_\beta \leftarrow E(\beta)$
    $s_0^\beta \sim s_0$ // initial state distribution
    for $t = 0, 1, \ldots, N$ **do**
      // execute current policy on the sampled robot and store the transition tuple to replay buffer
      $a_t^\beta \sim \pi(\cdot \mid s_t^\beta)$
      $(s_{t+1}^\beta, r_t^\beta) \sim M_\beta(\cdot \mid s_t^\beta, a_t^\beta)$
      // local reward shaping
      $r_t'^\beta \leftarrow r_t^\beta \cdot \exp(h \cdot \beta)$
      $\mathcal{R} \leftarrow \mathcal{R} \cup \{(s_t^\beta, a_t^\beta, s_{t+1}^\beta, r_t'^\beta)\}$
      sample $\{(s, a, s', r')\} \sim \mathcal{R}$
      train π with $\{(s, a, s', r')\}$ using RL
    end for
  end for
  // progress evolution parameter
  $\alpha \leftarrow \alpha + l_k$
  // clean up replay buffer
  $\mathcal{R} \leftarrow \{(s_t^\beta, a_t^\beta, s_{t+1}^\beta, r_t'^\beta) \in \mathcal{R}, \forall \beta \in [\alpha, \alpha + \xi]\}$
end while
return π

Local Randomized Evolution Progression We propose a randomized approach to address the above problem. At phase $k + 1$, instead of repetitively choosing a deterministic progressed α_{k+1}, we uniformly sample progressed evolution parameter β from a local neighborhood $[\alpha_k, \alpha_k + \xi]$ where $\xi \in \mathbb{R}^+$ and $\xi > l_k$, and train policy on the rollouts of robot $M_\beta = E(\beta)$. The optimization objective in Equation (3) is updated to be

$$\pi_{E(\alpha_{k+1})} = \arg\max_\pi \mathop{\mathbb{E}}_{\substack{\beta \sim U(\alpha_k, \alpha_k + \xi) \\ M_\beta = E(\beta)}} \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim M_\beta(\cdot|s_t, a_t)}} \sum_t \gamma^t r_t \tag{4}$$

where $U(p, q)$ denotes the uniform distribution over $[p, q] \subset \mathbb{R}$. The above randomized progression strategy allows the policy to be trained on sampled robots with small evolution to maintain sufficient sample efficiency and ensure adaptation, while also giving the policy a chance to risk on the robots with large evolution to improve the efficiency of policy transfer. Note that we choose the neighborhood size ξ to be larger than the progression step size l_k, which enables the policy to experience and explore more evolved robots with probability in advance. As shown in Section 5, the local randomized progression strategy improves the stability of training and achieves significantly higher performance.

We point out that the similar idea of domain randomization, i.e. fine-tuning neural networks on randomized domains, can also be seen in the literatures on Sim2Real domain transfer such as (Tobin et al., 2017) and (Sadeghi & Levine, 2016). Our robot-to-robot evolution approach can be also viewed as a series of mini robot domain randomization where the domain window $[\alpha_k, \alpha_k + \xi]$ is gradually shifting from the source robot $E(0)$ towards the target robot $E(1)$.

4.3. Evolution Reward Shaping

During local randomized evolution, to better adapt the policy towards the goal of the target robot, it is helpful to encourage the policy to give more weight to robots with larger α. To implement this, we devise a strategy to reshape the reward by making it a function of the evolution parameter α. Specifically, we scale the reward r_t received from rollouts on robot α to be

$$r_t' = r_t \cdot \exp(h \cdot \alpha) \tag{5}$$

where the evolution reward shaping factor $h \in \mathbb{R}_{\geq 0}$ controls the weight applied to the reward. The optimization objective in Equation (4) is then updated to be

$$\pi_{E(\alpha_{k+1})} = \arg\max_\pi \mathop{\mathbb{E}}_{\substack{\beta \sim U(\alpha_k, \alpha_k + \xi) \\ M_\beta = E(\beta)}} \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim M_\beta(\cdot|s_t, a_t)}} \sum_t \gamma^t r_t' \tag{6}$$

How should one use the h to control the weight on more evolved robots in practice? Under reasonable assumptions, we show the relation between the evolution reward shaping factor h and the resulted change of optimization objective with the following theorem.

Theorem 4.1. *Suppose the policy that optimizes the objective in Equation (6) with evolution reward shaping factor of h is the optimal policy $\pi^*_{M_\varphi}$ on robot $M_\varphi = E(\varphi), \varphi \in [\alpha_k, \alpha_k + \xi]$, i.e.*

$$\begin{aligned} &\arg\max_\pi \mathop{\mathbb{E}}_{\substack{\beta \sim U(\alpha_k, \alpha_k + \xi) \\ M_\beta = E(\beta)}} \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim M_\beta(\cdot|s_t, a_t)}} \sum_t \gamma^t r_t \exp(h \cdot \beta) \\ &= \pi^*_{M_\varphi} = \arg\max_\pi \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim M_\varphi(\cdot|s_t, a_t)}} \sum_t \gamma^t r_t \end{aligned} \tag{7}$$

Then when $\xi \to 0$, $\varphi = \alpha_k + \frac{1}{2}\xi + \frac{1}{4}h\xi^2 + o(\xi^2)$.

The proof of Theorem 4.1 is in the Section A. Theorem 4.1 shows that a positive evolution reward shaping factor shifts the objective of policy optimization towards the direction of the target robot compared to setting $h = 0$. This allows the policy to give more weight to the experiences gained with "more evolved" robots, without sacrificing sample efficiency due to changes in the sampling distribution.

4.4. Other Implementation Details

Adaptive Training Scheduling From the description so far, the training scheduling of robot evolution parameters α_k has been fixed. Moreover, the number of epochs trained in each evolution phase (i.e. N_e in Algorithm 1) is also fixed. In practice, however, we can dynamically schedule the training by changing both hyperparameters on the fly, especially when the difficulty of each transfer phase is different. Determining the progression step size l_k or the next α_k is usually hard since the training results are usually not predictable. A more practical strategy is to fix all l_k while setting the initial value of N_e to be small in each phase. When the training of policy struggles during the current phase, e.g. the reward or success rate drops significantly compared to previous phases, we iteratively increase N_e until the policy performance is sufficient to move on to the next phase. We adopted this strategy in the experiments in Section 5.2.

Replay Buffer Cleaning As the policy optimization moves on to the next phase, the past transition sampled from less evolved robots in previous phases are outdated and should no longer be used. To implement this, we remove transition tuples that are older than the current interpolation range from the replay buffer upon entering the next phase.

5. Experiments

The design of our REvolveR framework is motivated by the hypothesis that compared to directly transferring the policy from source to target robot, transferring the policy through a sequence of micro-evolutionary changes of robot dynamics is an easier task and achieves better sample efficiency and performance. To show this, we apply our REvolveR to two sets of robotic control tasks: MuJoCo Gym environments (Brockman et al., 2016), and Hand Manipulation Suite (Rajeswaran et al., 2018). We compare the performance to a variety of baselines including training policy from scratch, direct policy transfer, and imitation learning methods.

5.1. MuJoCo Gym Environments

Environments and Rewards We adopt the default Ant-v2 and Humanoid-v2 robots from MuJoCo Gym (Brockman et al., 2016) as our source robots. We construct three environments where the target robots are created by continuously modifying some properties of the source robots. In `Ant-length-mass` and `Humanoid-length-mass` environments, the mass and lengths of all leg bodies are changed; In `Ant-leg-emerge` environment, new legs and joints grow from the tip of the toe. The robot evolution is illustrated Figures 1 and 3(a)(c)(e).

Reward Function In all three environments, the robot agents get rewarded by the distance they moved forward. Specially, in `Humanoid-length-mass` environment, the robot agents are heavily penalized for falling down. The reward function is the same for source, target and all intermediate robots.

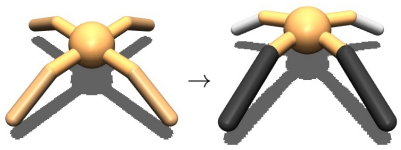
RL Algorithms We use two state-of-the-art actor-critic reinforcement learning algorithms, TD3 (Fujimoto et al., 2018) and SAC (Haarnoja et al., 2018), in our experiments. We first train both RL algorithms on the source robots until convergence and use the well-trained policy as the expert policy to be transferred to target robot. During transfer, both the actor and critic are updated. Note that the expert policy performance will be different for the two source robots because `Ant-leg-emerge` robots have an additional leg and therefore different state space.

Baselines We compare our method with the following baselines for learning a policy on the target robot.

- *From Scratch*: we train policy on the target robot from scratch with the RL algorithm.
- *Direct Transfer*: we initialize the target robot policy with the expert policy on source robot and fine-tune the policy directly on the target robot.
- *State-only Imitation Learning (SOIL) (Radosavovic et al., 2020)*: SOIL is an imitation learning method. It trains an inverse dynamics model to match the distribution of the next state between the student and teacher agents. Then it augments the policy gradient with a term that aims to maximize the probability of the predicted actions from inverse dynamics. In our experiments, the source robot is the teacher and the target robot is the student.

The above baseline methods are trained on the target robots for three million RL iterations on the `Ant-leg-emerge` and `Humanoid-length-mass` tasks and one million iterations on the `Ant-length-mass` task. We train our REvolveR for the same RL iterations as the baselines in total, not only on the target robot but on all intermediate robots during evolution. All methods are trained with five different random seeds. The experiment results are illustrated in Tables 3(b)(d)(f) [1].

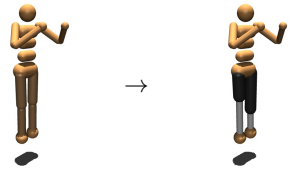
[1] We did not use the standard bold-line/shaded-area curves to illustrate the policy performance, because unlike ordinary RL algorithms that are trained on a single robot and have performance vs. time results, our REvolveR is only able to deliver **valid target robot performance** at the end of the policy transfer.



(a) `Ant-length-mass` environment.

	TD3	SAC
Expert on Source	6826.52	6985.90
From Scratch	4644.09 ± 502.05	5908.51 ± 339.81
Direct Transfer	4903.77 ± 801.12	6194.55 ± 165.82
SOIL	4891.67 ± 819.18	6061.32 ± 1102.58
Ours	**5903.28 ± 416.07**	**6473.21 ± 207.98**

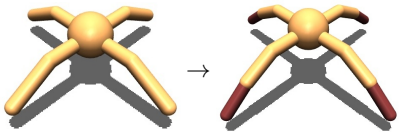
(b) `Ant-length-mass` policy transfer experiment results.



(c) `Humanoid-length-mass` environment.

	TD3	SAC
Expert on Source	6663.25	8271.70
From Scratch	5824.07 ± 233.46	6468.60 ± 157.26
Direct Transfer	6256.15 ± 253.63	7639.61 ± 278.02
SOIL	6414.25 ± 505.79	6970.15 ± 659.32
Ours	**7386.44 ± 151.24**	**7986.97 ± 129.21**

(d) `Humanoid-length-mass` policy transfer experiment results.



(e) `Ant-leg-emerge` environment.

	TD3	SAC
Expert on Source	6591.832	6431.32
From Scratch	2551.30 ± 316.95	3845.69 ± 243.92
Direct Transfer	3579.40 ± 118.06	6031.27 ± 320.81
SOIL	1703.62 ± 351.83	4533.09 ± 578.32
Ours	**4688.68 ± 270.51**	**6612.78 ± 264.50**

(f) `Ant-leg-emerge` policy transfer experiment results.

Figure 3. **Experiments on source-to-target policy transfer on the MuJoCo Gym environments.** All methods are trained for three million iterations with five different seeds. Mean and standard deviation of the reward of an epoch are reported. Our approach outperforms the baselines across different policy optimization schemes and across environments.

Results and Analysis Our REvolveR framework outperforms all related baselines by a notable margin in terms of episode reward, especially on `Ant-leg-emerge` and `Humanoid-length-mass`.

On `Ant-leg-emerge` environment, an interesting finding is that the performance of SOIL is even worse than directly transferring the policy. An explanation is that the emerging legs of the source robot have length and mass close to zero and show random behaviors with expert policy. Though the random behavior does not affect the source robot, it causes it to struggle at the start of training when directly transferred to target robot.

On `Humanoid-length-mass` environment, the expert policy is able to control the source humanoid robot to both stand and jog for higher rewards. However, due to dynamics mismatch, source expert policy cannot support target humanoid robot to stand. When directly trained on the target robot, even with source expert policy provided, all the baseline methods learned to discard jogging to learn standing first due to heavy penalty on falling down. As a comparison, when transferring the policy through continuously evolving intermediate robots with our REvolveR, both standing and jogging skills can be kept and smoothly transferred, which highlights the advantage of our method.

5.2. Hand Manipulation Suite

Robot Evolution We adopt the five-finger dexterous hand provided in the ADROIT platform (Kumar et al., 2013) as our source robot and follow Rajeswaran et al. (2018) for the environment settings. The robot evolve to the target robot of a two-finger gripper by gradually shrinking three fingers except the thumb and index finger. The robot evolution is illustrated Figures 1 and 4.

Task and Reward Function We use the three tasks from the proposed suite in (Rajeswaran et al., 2018): `Hammer`, `Relocate` and `Door`. In `Hammer`, the task is to pick up the hammer and smash the nail into the board; in `Relocate`, the task is to pick up the ball and take it to the target position; in `Door`, the task is to turn the door handle and fully open the door. In a sparse reward setting, only task completion is rewarded. In a dense reward setting, a distance reward is provided at every step.

The baselines compared against include *From Scratch* and *Direct Transfer* from Section 5.1. We also compare against DAPG (Rajeswaran et al., 2018) which is a variant of NPG (Rajeswaran et al., 2017) with demonstration-augmented policy gradient for learning from human demonstrations. The expert policies for the five-finger source robot are imported from (Rajeswaran et al., 2018) and used in all meth-

	Dense Reward	Sparse Reward
From Scratch	>100K	∞
Direct Finetune	>100K	∞
DAPG	17.1K	∞
Ours	-	**11.9K**

(a) `Hammer` task experiment results.

	Dense Reward	Sparse Reward
From Scratch	>100K	∞
Direct Finetune	43.5K	∞
DAPG	23.3K	∞
Ours	-	**18.1K**

(b) `Relocate` task experiment results.

	Dense Reward	Sparse Reward
From Scratch	-	∞
Direct Finetune	7.6K	∞
DAPG	5.4K	∞
Ours	-	**2.6K**

(c) `Door` task experiment results.

Table 1. **Experiments on the Hand Manipulation Suite**. The evaluation metrics is the number of epochs needed to reach 90% task success rate. Our method with sparse reward outperforms all the baselines even with dense reward.

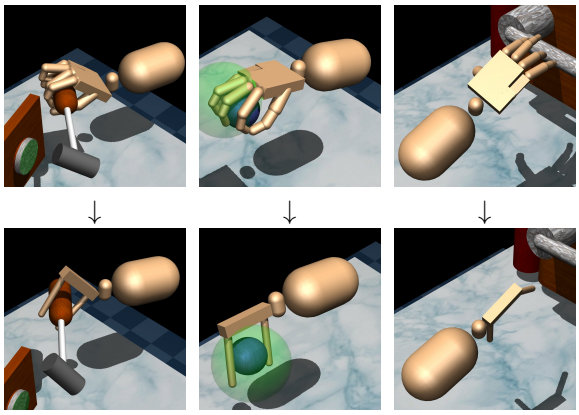
Randomized Evolution	Reward Shaping Factor	Reward
✗	$h = 0.0$	5363.89 ± 419.90
✓	$h = 0.0$	5844.46 ± 33.63
✓	$h = 1.0$	6279.35 ± 290.33

Table 2. **Ablation studies** on local randomized evolution and evolution reward shaping. The RL algorithm used in the experiments is SAC and the task evaluated is `Ant-length-mass`.

ods as needed.

For our REvolveR, we use NPG (Rajeswaran et al., 2017) as our RL algorithm. We use the adaptive training scheduling strategy proposed in Section 4.4 to improve training efficiency. Therefore, the total number of RL iterations cannot be set beforehand to fairly compare the performance under the same number of iterations. So we instead compare in terms of the number of RL optimization steps needed to reach 90% success rate on the tasks. The results are illustrated in Table 1.

Results and Analysis In sparse reward case, all baselines never receive positive reward for improving itself and are ineffective in solving the tasks. This is because due to dynamics mismatch, source expert policy cannot find a single successful trajectory on the target robot, e.g. cannot pickup the ball or hammer. At the same time, the exploration in the high-dimension is too hard. As a comparison, when transferring the policy through continuously evolving intermediate robots with REvolveR, the intermediate robots maintain sufficient success rate to ensure sample efficiency



(a) `Hammer`. (b) `Relocate`. (c) `Door`.

Figure 4. **Hand Manipulation Suite tasks.** (a) `Hammer` environment: the goal is to pick up the hammer and smash the nail into the board; (b) `Relocate` environment: the goal is to pick up the blue ball and take it to the desired site shown by the green semi-transparent sphere; (c) `Door` environment: the goal is to switch the latch and fully open the door.

and successfully transfer the policy.

We show the detailed results of comparison in Table 1. As the results show, when trained with sparse reward, our REvolveR even outperforms baselines trained with dense reward in terms of total number of iterations. This again highlights the advantage of our method in terms of improving sample efficiency and performance in policy transfer.

5.3. Ablation Studies

We perform ablation experiments on `Ant-length-mass` environment. The RL algorithm used in the experiments is SAC (Haarnoja et al., 2018). We ablate the following two components of our method:

Deterministic vs. Local Randomized Evolution We study the effect of using local randomized evolution progression strategy proposed in Equation (4) and compare against deterministic evolution in Equation (3). As illustrated in Table 2, local randomized evolution progression not only improves performance of transfer but also improves the robustness as shown by the decrease of standard deviation of the episode reward.

Evolution Reward Shaping We study the effect of evolution reward shaping as proposed in Equation (5) and compare against Equation (4) without reward shaping. As illustrated in Table 2, evolution reward shaping can effectively improve performance as shown by the improvement of the mean of the episode reward.

6. Conclusion

In this paper, we propose a novel method named REvolveR for robotic policy transfer between two different robots so that one does not have to train a policy from scratch for every new robot. Our method is based on continuous evolutionary models implemented in a physics simulator. An expert policy on the source robot can be transferred through training on a sequence of intermediate robots that evolve into the target robot. We conduct experiments on several tasks on MuJoCo simulation engine and show that the proposed method can effectively transfer the policy across robots and achieve superior sample efficiency on new robots and is especially advantageous in sparse reward settings.

Acknowledgement

This work is in part funded by JST AIP Acceleration, Grant Number JPMJCR20U1, Japan. DP was supported by NSF IIS-2024594.

References

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Chen, T., Murali, A., and Gupta, A. Hardware conditioned policies for multi-robot transfer learning. In *Advances in Neural Information Processing Systems*, pp. 9355–9366, 2018.

Cheney, N., MacCurdy, R., Clune, J., and Lipson, H. Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. *ACM SIGEVOlution*, 2014.

Clune, J., Pennock, R. T., Ofria, C., and Lenski, R. E. Ontogeny tends to recapitulate phylogeny in digital organisms. *The American Naturalist*, 180(3):E54–E63, 2012.

Coumans, E. and Bai, Y. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016.

Daudelin, J., Jing, G., Tosun, T., Yim, M., Kress-Gazit, H., and Campbell, M. An integrated system for perception-driven autonomy with modular robots. *Science Robotics*, 2018.

Fujimoto, S., Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pp. 1587–1596. PMLR, 2018.

Gilpin, K., Kotay, K., Rus, D., and Vasilescu, I. Miche: Modular shape formation by self-disassembly. *IJRR*, 2008.

Gupta, A., Savarese, S., Ganguli, S., and Fei-Fei, L. Embodied intelligence via learning and evolution. *arXiv preprint arXiv:2102.02202*, 2021.

Ha, D. Reinforcement learning for improving agent design. *arXiv preprint arXiv:1810.03779*, 2018.

Ha, S., Coros, S., Alspach, A., Kim, J., and Yamane, K. Joint optimization of robot design and motion parameters using the implicit function theorem. In *Robotics: Science and Systems*, 2017.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. 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.

Hejna, D., Pinto, L., and Abbeel, P. Hierarchically decoupled imitation for morphological transfer. In *International Conference on Machine Learning*, pp. 4159–4171. PMLR, 2020.

Hejna III, D. J., Abbeel, P., and Pinto, L. Task-agnostic morphology evolution. *arXiv preprint arXiv:2102.13100*, 2021.

Ho, J. and Ermon, S. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29:4565–4573, 2016.

Huang, W., Mordatch, I., and Pathak, D. One policy to control them all: Shared modular policies for agent-agnostic control. In *International Conference on Machine Learning*, pp. 4455–4464. PMLR, 2020.

Kriegman, S., Cheney, N., and Bongard, J. How morphological development can guide evolution. *Scientific reports*, 8(1):1–10, 2018.

Kumar, V., Xu, Z., and Todorov, E. Fast, strong and compliant pneumatic actuation for dexterous tendon-driven hands. In *2013 IEEE international conference on robotics and automation*, pp. 1512–1519. IEEE, 2013.

Kurin, V., Igl, M., Rocktäschel, T., Boehmer, W., and Whiteson, S. My body is a cage: the role of morphology in graph-based incompatible control. *arXiv preprint arXiv:2010.01856*, 2020.

Liu, F., Ling, Z., Mu, T., and Su, H. State alignment-based imitation learning. *arXiv preprint arXiv:1911.10947*, 2019.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

Luo, Y., Xu, H., Li, Y., Tian, Y., Darrell, T., and Ma, T. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. In *International Conference on Learning Representations*, 2018.

Murata, S. and Kurokawa, H. Self-reconfigurable robots. *IEEE Robotics & Automation Magazine*, 2007.

Ng, A. Y., Russell, S. J., et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, pp. 2, 2000.

Pan, X., Garg, A., Anandkumar, A., and Zhu, Y. Emergent hand morphology and control from optimizing robust grasps of diverse objects. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7540–7547. IEEE, 2021.

Pathak, D., Lu, C., Darrell, T., Isola, P., and Efros, A. A. Learning to control self-assembling morphologies: a study of generalization via modularity. *NeurIPS*, 2019.

Radosavovic, I., Wang, X., Pinto, L., and Malik, J. State-only imitation learning for dexterous manipulation. *arXiv preprint arXiv:2004.04650*, 2020.

Rajeswaran, A., Lowrey, K., Todorov, E., and Kakade, S. Towards generalization and simplicity in continuous control. *arXiv preprint arXiv:1703.02660*, 2017.

Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*, Pittsburgh, Pennsylvania, June 2018. doi: 10.15607/RSS.2018.XIV.049.

Romanishin, J. W., Gilpin, K., and Rus, D. M-blocks: Momentum-driven, magnetic modular robots. In *IROS*, 2013.

Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Sadeghi, F. and Levine, S. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Network*, 2009.

Schaff, C., Yunis, D., Chakrabarti, A., and Walter, M. R. Jointly learning to construct and control agents using deep reinforcement learning. *arXiv preprint arXiv:1801.01432*, 2018.

Sims, K. Evolving virtual creatures. In *Computer graphics and interactive techniques*, 1994a.

Sims, K. Evolving 3d morphology and behavior by competition. *Artificial life*, 1994b.

Stoy, K., Brandt, D., Christensen, D. J., and Brandt, D. *Self-reconfigurable robots: an introduction*. Mit Press Cambridge, 2010.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 23–30. IEEE, 2017.

Von Neumann, J., Burks, A. W., et al. Theory of self-reproducing automata. *IEEE Transactions on Neural Networks*, 1966.

Wang, R., Lehman, J., Clune, J., and Stanley, K. O. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019a.

Wang, T., Liao, R., Ba, J., and Fidler, S. Nervenet: Learning structured policy with graph neural networks. *ICLR*, 2018.

Wang, T., Zhou, Y., Fidler, S., and Ba, J. Neural graph evolution: Towards efficient automatic robot design. *arXiv preprint arXiv:1906.05370*, 2019b.

Whitman, J., Bhirangi, R., Travers, M., and Choset, H. Modular robot design synthesis with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 10418–10425, 2020.

Whitman, J., Travers, M., and Choset, H. Learning modular robot control policies. *arXiv preprint arXiv:2105.10049*, 2021.

Wright, C., Johnson, A., Peck, A., McCord, Z., Naaktgeboren, A., Gianfortoni, P., Gonzalez-Rivero, M., Hatton, R., and Choset, H. Design of a modular snake robot. In *IROS*, 2007.

Yim, M., Duff, D. G., and Roufas, K. D. Polybot: a modular reconfigurable robot. In *ICRA*, 2000.

Zhao, A., Xu, J., Konaković-Luković, M., Hughes, J., Spielberg, A., Rus, D., and Matusik, W. Robogrammar: graph grammar for terrain-optimized robot design. *ACM Transactions on Graphics (TOG)*, 39(6):1–16, 2020.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

A. Proof of Theorem 4.1

Since $E(\cdot)$ is a continuous function of robot models, we assume the transition dynamics of the robot $E(\alpha)$ is differentiable and locally L_1-Lipschitz w.r.t. α in the sense that

$$\exists \varepsilon > 0, ||E(\alpha)(s,a) - E(\alpha')(s,a)|| \leq L_1|\alpha - \alpha'|, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}, \forall |\alpha' - \alpha| < \varepsilon \tag{8}$$

Moreover, we follow (Luo et al., 2018) and assume the value functions of the robot models are L_2-Lipschitz w.r.t to some norm $||\cdot||$ in state space in the sense that

$$|V^{\pi,M}(s) - V^{\pi,M}(s')| \leq L_2||s - s'||, \forall s, s' \in \mathcal{S} \tag{9}$$

By the assumption in Equation (9) that the value functions of the robots are L_2-Lipschitz, as proven in Lemma 4.1 of (Luo et al., 2018), $\forall \phi > 0$, $M = E(\alpha)$, $M' = E(\alpha + \phi)$, we have

$$\begin{aligned}
\exists \varepsilon > 0, \text{s.t. } \mathop{\mathbb{E}}_{s}[|V^{\pi,M}(s) - V^{\pi,M'}(s)|] &\leq \frac{\gamma}{1-\gamma} L_2 \mathop{\mathbb{E}}_{(s,a)\sim\pi} ||M(s,a) - M'(s,a)|| \\
&= \frac{\gamma}{1-\gamma} L_2 \mathop{\mathbb{E}}_{(s,a)\sim\pi} ||E(\alpha)(s,a) - E(\alpha')(s,a)|| \\
&\leq \frac{\gamma}{1-\gamma} L_2 L_1 |\alpha - \alpha'|, \forall |\alpha' - \alpha| < \varepsilon
\end{aligned} \tag{10}$$

Since the value functions are differentiable by assumption in Equation (8), suppose $D = |\frac{\partial V^{\pi,E(\alpha)}}{\alpha}|$ is the absolute value of the derivative of $V^{\pi,E(\alpha)}$ w.r.t. to α. According to Equation (10), D is bounded. We make a (strong) assumption that for all $\varphi \in [\alpha, \alpha + \xi]$, policy $\pi^*_{E(\varphi)}$ only achieves the best expected reward on robot $E(\varphi)$, so that

$$\forall \varphi, \beta \in [\alpha, \alpha + \xi], V^{\pi^*_{E(\varphi)}, E(\varphi)} - V^{\pi^*_{E(\varphi)}, E(\beta)} = D \cdot |\varphi - \beta| + o(|\beta - \varphi|^2) = D \cdot |\varphi - \beta| + o(\xi^2) \tag{11}$$

From the definition, the value function of a policy π on uniformly sampled robots $E(\beta)$ from $\beta \sim U(\alpha, \alpha + \xi)$ is

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\beta\sim U(\alpha,\alpha+\xi)} \mathop{\mathbb{E}}_{(s_t,a_t)\sim\pi,E(\beta)} \sum_t \gamma^t r_t \cdot \exp(h\beta) \\
=&\mathop{\mathbb{E}}_{\beta\sim U(\alpha,\alpha+\xi)} \exp(h\beta) \mathop{\mathbb{E}}_{(s_t,a_t)\sim\pi,E(\beta)} \sum_t \gamma^t r_t \\
=&\mathop{\mathbb{E}}_{\beta\sim U(\alpha,\alpha+\xi)} \exp(h\beta) V^{\pi,E(\beta)} \\
=&\frac{1}{\xi} \int_{\beta=\alpha}^{\alpha+\xi} \exp(h\beta) V^{\pi,E(\beta)} \, \mathrm{d}\beta
\end{aligned} \tag{12}$$

Supposed the π that optimizes Equation (12) is the policy that directly optimizes on one robot $E(\varphi), \varphi \in [\alpha, \alpha + \xi]$, i.e. $\pi^*_{E(\varphi)}$ optimizes Equation (12), then we treat $\pi^*_{E(\varphi)}$ and $V^{\pi^*_{E(\varphi)}, E(\varphi)}$ as constants and the following variation should be zero

$$\delta \int_{\beta=\alpha}^{\alpha+\xi} \exp(h\beta)(V^{\pi^*_{E(\varphi)}, E(\beta)} - V^{\pi^*_{E(\varphi)}, E(\varphi)}) \, \mathrm{d}\beta = 0 \tag{13}$$

which means

$$\begin{aligned}
0 =& \int_{\beta=\alpha}^{\alpha+\xi} \frac{\partial}{\partial\beta} [\exp(h\beta)(V^{\pi^*_{E(\varphi)}, E(\beta)} - V^{\pi^*_{E(\varphi)}, E(\varphi)})] \, \mathrm{d}\beta \\
\approx& \int_{\beta=\alpha}^{\varphi} \frac{\partial}{\partial\beta} [\exp(h\beta) D|\varphi - \beta|] \, \mathrm{d}\beta - \int_{\beta=\varphi}^{\alpha+\xi} \frac{\partial}{\partial\beta} [\exp(h\beta) D|\varphi - \beta|] \, \mathrm{d}\beta + o(\xi^2) \\
=& D \int_{\beta=\alpha}^{\varphi} \frac{\partial}{\partial\beta} [\exp(h\beta)(\varphi - \beta)] \, \mathrm{d}\beta - D \int_{\beta=\varphi}^{\alpha+\xi} \frac{\partial}{\partial\beta} [\exp(h\beta)(\beta - \varphi)] \, \mathrm{d}\beta + o(\xi^2) \\
=& \frac{e^{h\alpha}}{h} D[(h\varphi - 1)(2e^{h(\varphi-\alpha)} - 1 - e^{h\xi}) + \\
& h(\alpha + \xi)e^{h\xi} - e^{h\xi} - h\varphi e^{h(\varphi-\alpha)} + e^{h(\varphi-\alpha)} - h\varphi e^{h(\varphi-\alpha)} + e^{h(\varphi-\alpha)} + h\alpha - 1] + o(\xi^2) \\
=& \frac{e^{h\alpha}}{2h} D[(\alpha h^2 \xi^2 + 2\alpha h\xi + 4\alpha + 2h\xi^2 + 2\xi) - (h^2\xi^2 + 2h\xi + 4)\varphi] + o(\xi^2)
\end{aligned} \tag{14}$$

The last equation assumes $0 \leq h(\varphi - \alpha) \leq h\xi \ll 1$ and used second-order Taylor approximation of $e^y = 1 + y + \frac{1}{2}y^2 + o(y^2), y \in \mathbb{R}$. Then we have

$$\begin{aligned} \varphi &= \alpha + \frac{1}{2}\xi + \frac{\xi^2 h(2 - \xi h)}{8 + 4\xi h + 2\xi^2 h^2} + o(\xi^2) \\ &= \alpha + \frac{1}{2}\xi + \frac{1}{4}h\xi^2 + o(\xi^2) \end{aligned} \tag{15}$$

Therefore, if

$$\begin{aligned} &\lim_{\xi \to 0} \left[\arg\max_{\pi} \mathop{\mathbb{E}}_{\substack{\beta \sim U(\alpha, \alpha+\xi) \\ M_\beta = E(\beta)}} \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim M_\beta(\cdot|s_t, a_t)}} \sum_t \gamma^t r_t \exp(h\beta) \right] \\ &= \pi^*_{M_\varphi} = \arg\max_{\pi} \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim M_\varphi(\cdot|s_t, a_t)}} \sum_t \gamma^t r_t, \end{aligned} \tag{16}$$

then when $\xi \to 0$, $\varphi = \alpha + \frac{1}{2}\xi + \frac{1}{4}h\xi^2 + o(\xi^2)$. □

B. Experiment Details

Caching Intermediate Robots In practice, frequently calling the function for generating simulation environments could be costly. To avoid repetitively generating a new environment at the start of every epoch and to speed up the training process, we pre-generate and cache a large number (e.g. 1,000) of environments. At the start of each epoch, we randomly sample within the desired interpolation range and fetch a simulation environment from the cache. When the number of cached environments is large, the above sampling behavior is a good approximation to sampling from continuous robot evolution.

Experiment Hyperparameter Setting We illustrate the hyperparameters of the neural networks used in Gym and Hand Manipulation Suite experiments, including layer size, batch size and learning rate, in Table 3.

	MuJoCo Gym Environments	Hand Manipulation Suite
Actor	$[s, 256, 256, a]$	$[s, 32, 32, a]$
Critic	$[s + a, 256, 256, 1]$	$[s + a, 32, 32, 1]$
Batch size	256	16
Learning Rate	3×10^{-4}	1×10^{-4}

Table 3. Size of the Neural network used in the experiments. s and a represents the dimension of state space and action space respectively.

C. Qualitative Results

We show the process of transferring policy on intermediate robots on Hand Manipulation Suite tasks in Figure 5. We show the transferred policy on the target robot of Hand Manipulation Suite tasks in Figure 6. For more details, please refer to the supplementary video.

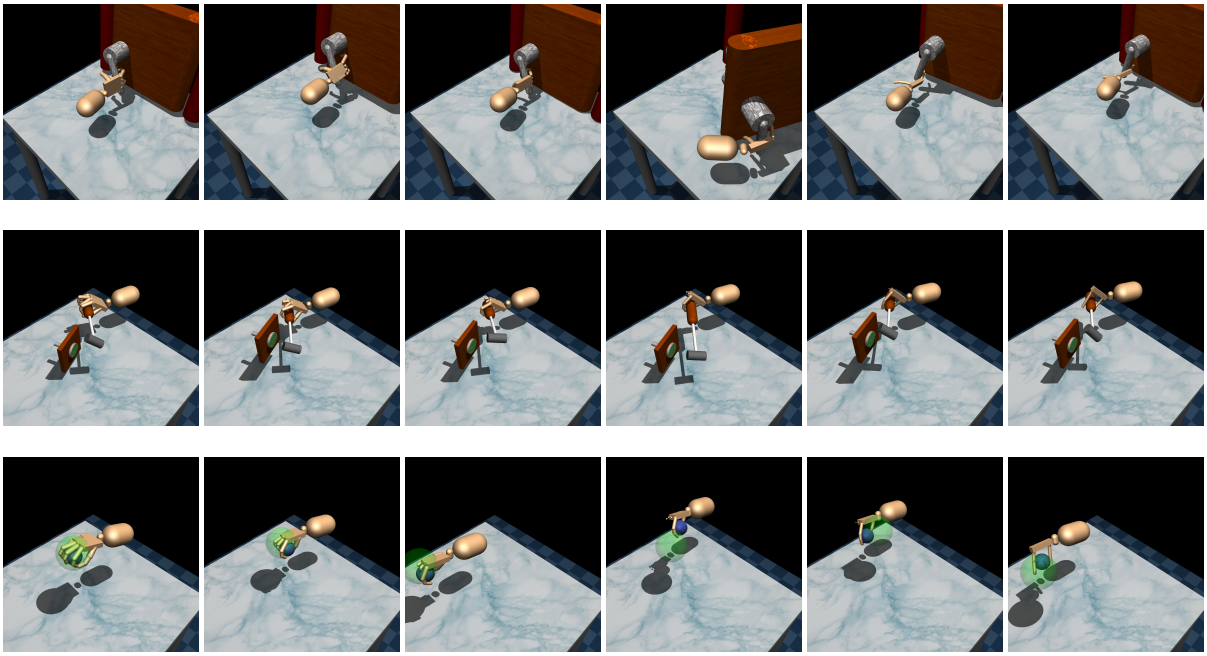


Figure 5. **Visualization of policy on evolving intermediate robots on Hand Manipulation Suite tasks.** The three rows shows `Door`, `Hammer`, and `Relocate` tasks respectively. From left to right in each row, we show a snapshot of robot at evolution parameters α at $0, 0.2, 0.4, 0.6, 0.8, 1$ respectively in the six columns.

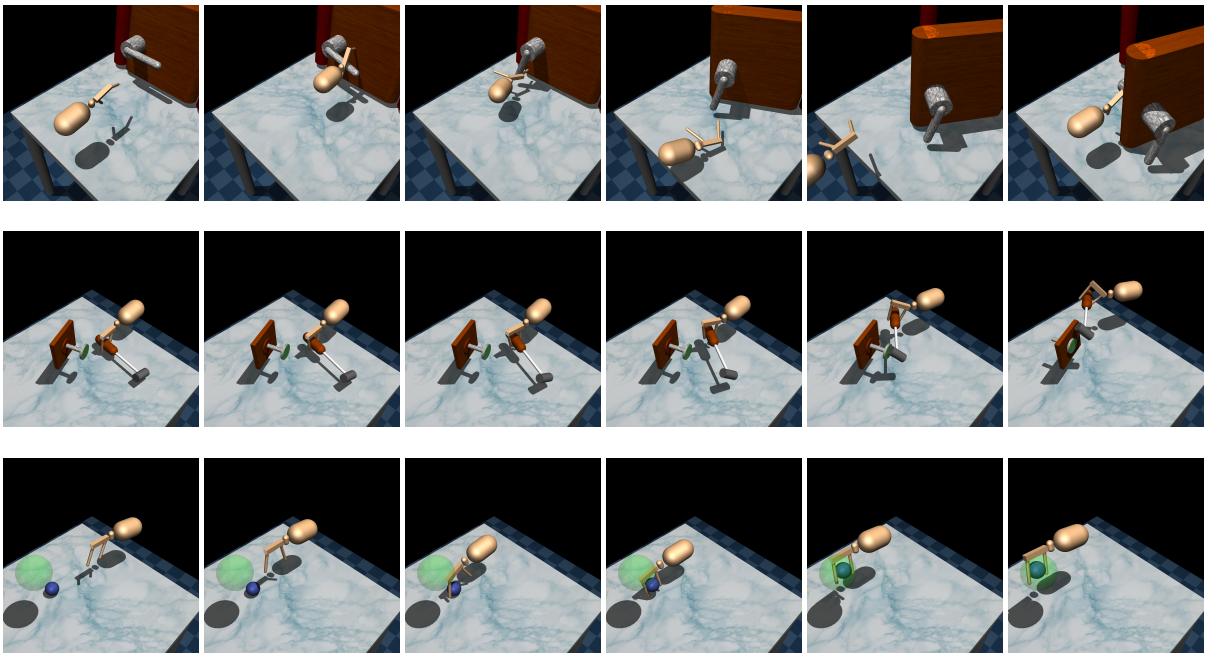


Figure 6. **Qualitative results on the target robot on Hand Manipulation Suite tasks.** We show the transferred policy on target robots. The three rows shows `Door`, `Hammer`, and `Relocate` tasks respectively. From left to right in each row is a policy rollout on target robot at $\alpha = 1$.