From Demonstrations to Adaptations: Assessing Imitation Learning Robustness and Learned Reward Transferability

Nathan Van Utrecht, Cody Fleming

May 2025

Abstract

Adversarial Inverse Reinforcement Learning (AIRL) aims to learn disentangled reward functions from expert demonstrations, holding theoretical promise for robust policy generalization and effective reward transfer to novel environments. This work provides a systematic empirical comparison of AIRL against direct imitation methods (Behavioral Cloning and GAIL) focusing on these aspects. We evaluate policies learned on four MuJoCo benchmarks and their zero-shot performance under various environmental perturbations. Furthermore, we assess the utility of AIRL-learned rewards for training new SAC agents in these modified settings. Results show that AIRL policies often demonstrate better zero-shot robustness than BC and GAIL. Critically, while AIRL's transferred reward can enable SAC agents to achieve near-oracle performance in certain modified dynamics (e.g., Hopper), its effectiveness significantly diminishes when faced with substantial task changes (e.g., Pusher goal shifts) or morphological alterations (e.g., Ant disabled leg), especially if such variations were unobserved during AIRL's training. Our findings underscore both the potential of AIRL for transfer and the critical need for strategies that enhance the generalization of learned reward functions from limited demonstrations. All code can be found at `https://github.com/nvan21/Honors-Capstone-IL-Robustness/tree/main`.

1 Introduction

Reinforcement Learning (RL) has achieved remarkable successes in domains ranging from game playing to continuous-control robotics, by framing decision-making as a trial-and-error process in which an agent maximizes a hand-designed

reward function [7]. However, crafting dense, informative rewards for complex real-world tasks (such as autonomous driving or surgical manipulation) remains a significant challenge. Poorly specified rewards can lead to undesirable behavior, reward hacking, or poor generalization when the environment deviates even slightly from training conditions [1].

Imitation Learning (IL) offers an alternative by directly learning from expert demonstrations, sidestepping explicit reward engineering. Methods like Behavioral Cloning (BC) [10] simply map states to expert actions via supervised learning, but suffer from covariate shift and brittle generalization under environmental perturbations. Generative Adversarial Imitation Learning (GAIL) [6] formulates the imitation problem as a two-player minimax game: a discriminator is trained to distinguish expert from agent state-action pairs, while the policy is updated via RL [14] to maximize the likelihood of fooling the discriminator. However, this approach typically requires large amounts of environment samples, can suffer from training instability, and may exhibit mode collapse [11] when expert demonstrations provide limited coverage of the state-action space.

In contrast, Inverse Reinforcement Learning (IRL) seeks to infer the underlying reward function that produced the expert demonstrations, rather than directly learning a policy. Adversarial Inverse Reinforcement Learning (AIRL) [3] combines adversarial training with IRL to recover a disentangled surrogate reward function that ideally captures the true task objectives and remains invariant to changes in environment dynamics. Once learning, this surrogate reward can be used to train new RL agents in modified environments, offering a pathway to both robustness and transferability.

However, despite AIRL's theoretical promise for reward transfer, there is a lack of systematic, empirical comparisons between IL and IRL approaches along two key dimensions: (i) the zero-shot robustness of learned policies when deployed in perturbed environments, and (ii) the effectiveness of AIRL-recovered rewards in training new agents under shifted dynamics or objectives. To fill this gap, we conduct a comparative study across four continuous-control benchmarks: Inverted Pendulum, Hopper, Pusher, and Ant. Each environment had its own set of perturbations that the algorithms were tested on. These will be explained in Section 3.

Our contributions are fourfold:

1. **Baseline Evaluation.** Evaluate BC, GAIL, and AIRL policies along with an Expert Soft Actor-Critic (SAC) [5] on the baseline versions of the above environments.

2. **Zero-Shot Robustness.** Measure performance degradation (without further training) of BC, GAIL, and AIRL policies in perturbed variants, identifying which approach offers the greatest robustness.

3. **Reward Transferability.** Use the AIRL-recovered reward function to train new SAC agents from scratch in each modified environment, assessing whether the surrogate reward encodes meaningful, generalizable objectives.

4. **Comparative Analysis.** Contrast zero-shot IL policies against SAC agents trained on transferred AIRL rewards, revealing conditions under which each method excels.

2 Background

This section surveys the algorithms under comparison: the two IL approaches (BC, GAIL), the IRL-based AIRL framework, and the use of transferred rewards in downstream SAC training. For notation, π_θ represents a policy learned by the algorithm, $\hat{\pi}_\theta$ represents the expert policy

2.1 Behavioral Cloning

Behavioral Cloning casts imitation as a supervised regression (continuous actions) or classification (discrete actions) problem. Given a fixed dataset of expert demonstrations $\{(s_t, a_t)\}_{t=1}^{N}$, BC minimizes an objective such as mean-squared error for continuous outputs, or cross-entropy for discrete actions. Since all of the environments tested in this paper are continuous, just the continuous formulation will be presented here:

$$\mathcal{L}_{BC}(\theta) = \frac{1}{N} \sum_{t=1}^{N} \left[\pi_\theta(a_t|s_t) - \pi_\theta(\hat{a_t}|s_t) \right]^2 \tag{1}$$

This formulation is efficient to train since it requires no environment interaction. However, because the policy only learns from fixed demonstration data, small prediction errors at test time can compound as the agent drifts into unfamiliar states, leading to degraded performance [10]. It is almost entirely dependent on the quality of the demonstration data (i.e. how good the expert is and how much of the state space is in the dataset).

2.2 Generative Adversarial Imitation Learning

Generative Adversarial Imitation Learning [6] improves on BC by framing imitation learning as an adversarial process, inspired by Generative Adversarial Networks (GANs) [4]. GAIL involves training two neural networks: a generator (the agent's policy, π_θ) and a discriminator (D_ϕ).

The generator aims to produce state-action pairs (or trajectories) that are indistinguishable from those generated by the expert policy. It interacts with the environment to generate those trajectories. The discriminator is trained to distinguish between state-action pairs sampled from the expert demonstration and those generated by the current policy. The discriminator outputs a probability $D_\phi(s, a) \in [0, 1]$ indicating the likelihood that the state-action pair comes from the expert.

The core idea is that the discriminator's output provides a learned reward signal for the policy. The policy is trained using a reinforcement learning algorithm (e.g. Trust Region Policy Optimization [12] or Proximal Policy Optimization [13]) to maximize the pseudo-reward $-\log(1-D_\phi(s, a))$. The policy is effectively trying to fool the discriminator into classifying its generated state-action pairs as expert-like. Concurrently, the discriminator is trained to minimize the binary cross-entropy loss for classifying expert versus generated samples. This leads to a min-max objective similar to GANs [4]:

$$\min_\theta \max_\phi \quad E_{\pi_\theta}[\log(D_\phi(s, a))] + E_{\hat{\pi}}[\log(1 - D_\phi(s, a))] - \lambda H(\pi_\theta) \tag{2}$$

where E_{π_θ} denotes the expectation over state-action pairs generated by policy π_θ, $E_{\hat{\pi}}$ denotes the expectation over expert state-action pairs, $H(\pi_\theta)$ is an optional entropy regularization term for the policy to encourage exploration, and λ is its weight.

In practice, the training alternates between updating the discriminator parameters ϕ using samples from both the expert dataset and the current policy, and updating the policy parameters θ using a policy gradient method with the rewards $r(s, a) = -\log(1-D_\phi(s, a))$ (or sometimes $\log(D_\phi(s, a))$). By interacting with the environment and receiving feedback from the dynamic discriminator, GAIL can learn to correct for the covariate shift problem that plagues BC [6]. However, GAIL can be less sample-efficient in terms of environment interactions than BC (though often more efficient in terms of expert data usage for complex tasks) and can suffer from the training instabilities common in adversarial learning.

2.3 Adversarial Inverse Reinforcement Learning

Adversarial Inverse Reinforcement Learning [3] builds upon the adversarial framework of GAIL but aims to explicitly recover a reward function from expert demonstrations in addition to directly learning a policy. By doing so, AIRL can learn reward functions that are more robust to changes in environment dynamics and potentially more transferable to different tasks.

Like GAIL, AIRL employs a generator (π_θ) and a discriminator (D_ϕ). However, the key innovation in AIRL lies in the specific structure of its discriminator and the interpretation of its learned components. The discriminator in AIRL is designed to not only distinguish expert trajectories from policy-generated trajectories but also to implicitly learn a reward function $r_\phi(s, a)$ that explains the expert's behavior.

The discriminator in AIRL typically takes the form:

$$D_\phi(s, a, s') = \frac{\exp(f_\phi(s, a, s'))}{\exp(f_\phi(s, a, s')) + \pi_\theta(a|s)} \tag{3}$$

where s' is the next state resulting from taking action a in state s, and $\pi_\theta(a|s)$ is the probability of taking action a in state s under the current policy. The function $f_\phi(s, a, s')$ is learned by the discriminator's network.

A crucial aspect of AIRL is that $f_\phi(s, a, s')$ is often decomposed to disentangle the reward function from the environment dynamics. A common formulation for f_ϕ is:

$$f_\phi(s, a, s') = g_\phi(s, a) + \gamma h_\phi(s') - h_\phi(s) \tag{4}$$

Here, $g_\phi(s, a)$ is interpreted as the learned reward function, $h_\phi(s)$ is a state-only potential function (often called a shaping function), and γ is the discount factor. This specific form helps AIRL learn a reward function $g_\phi(s, a)$ that is theoretically invariant to changes in environment dynamics [3], a property not explicitly guaranteed by GAIL. The policy π_θ is then trained using standard RL algorithms (e.g., SAC, PPO) with $g_\phi(s, a)$ as the reward.

State-Only Reward Formulation in AIRL

A notable, and often practically preferred, variant within the AIRL framework involves learning a state-only reward function, typically denoted $g_\phi(s)$. In this case, the reward an agent receives depends only on the state it is in, rather than the specific action taken to reach or in that state. The decomposition of f_ϕ is then modified such that the reward component g_ϕ is a function of the current

state s only:

$$f_\phi(s, a, s') = g_\phi(s) + \gamma h_\phi(s') - h_\phi(s) \tag{5}$$

The policy π_θ is subsequently trained to maximize the expected cumulative sum of these state-based rewards, i.e., using $r_t = g_\phi(s_t)$.

It is important to recognize that even when learning a state-only reward $g_\phi(s)$, the action a still influences the overall AIRL process: it determines the next state s' through the environment dynamics, and the policy's action probability $\pi_\theta(a|s)$ is a component of the discriminator's calculation (Equation 3). The *state-only* characteristic refers specifically to the input of the learned reward component g_ϕ.

This state-only reward formulation can offer several advantages. As observed by Fu et al. [3], using state-only rewards $g_\phi(s)$ can lead to more stable training and often yields better performance on complex continuous control tasks. This is potentially because state-action rewards ($g_\phi(s, a)$) have a higher capacity, making them more prone to overfitting, especially when the amount of expert demonstration data is limited. Learning a simpler, state-dependent reward function reduces the complexity of the IRL problem, which can improve sample efficiency and robustness against spurious correlations in the action space.

However, a state-only reward function may not be able to capture all nuances of expert behavior if optimality critically depends on specific action choices that are not solely distinguishable by the resulting state or the current state itself (e.g., if different actions in the same state lead to similar next states but should inherently have different rewards due to unmodeled factors like energy consumption or risk). Despite this potential limitation, for many tasks where the goal is primarily defined by reaching or maintaining certain state configurations, state-only rewards learned via AIRL provide a powerful and effective approach.

The overall training objective for AIRL, aiming to find a saddle point, remains:

$$\min_\theta \max_\phi \quad E_{\pi_\theta}[\log(D_\phi(s, a, s'))] + E_{\hat{\pi}}[\log(1 - D_\phi(s, a, s'))] - \lambda H(\pi_\theta) \tag{6}$$

The key difference lies in the parameterization of g_ϕ within f_ϕ, which determines whether the policy π_θ is optimized using $g_\phi(s, a)$ or $g_\phi(s)$.

By learning an explicit reward function (either state-action or state-only), AIRL aims to offer better interpretability and robustness compared to direct imitation methods like GAIL. The recovered reward function can also be used for downstream tasks or further refinement. However, like other adversarial methods, it can be sensitive to hyperparameter tuning and may require careful design of the network architectures for g_ϕ and h_ϕ.

3 Methods

This section details the experimental methodology employed to investigate the research questions laid out in Section 1. All implementation details, including network architectures and hyperparameters for the algorithms, are provided in Appendix A.

3.1 Algorithms Compared

The core algorithms evaluated in this study, previously introduced in Section 2, are:

1. **Behavioral Cloning (BC):** As described in Section 2.1, BC directly learns a state-to-action mapping via supervised learning.
2. **Generative Adversarial Imitation Learning (GAIL):** Detailed in Section 2.2, GAIL adversarially trains a policy to match the expert's state-action distribution.
3. **Adversarial Inverse Reinforcement Learning (AIRL):** As outlined in Section 2.3, AIRL aims to recover a disentangled reward function. For our experiments, we specifically utilize the state-only reward formulation ($g_\phi(s)$) of AIRL.
4. **Soft Actor-Critic with Transferred AIRL Reward (SAC-AIRL):** This involves training a Soft Actor-Critic (SAC) [5] agent using the reward function learned by AIRL in the base environment.
5. **Expert Soft Actor-Critic (Expert SAC):** As an upper-bound comparison, a SAC agent is trained in modified environments using the ground-truth reward.

3.2 Environments and Modifications

Experiments are conducted using four continuous control environments from the Gymnasium MujoCo suite [17] [16]: InvertedPendulum-v5, Hopper-v5, Pusher-v5, and Ant-v5. An image of each environment is displayed in Figure 1. For each of these base environments, we create several modified versions to evaluate policy robustness and reward transferability. These modifications, categorized as either dynammics changes or task changes, are detailed below:

- **InvertedPendulum-v5 Modifications**

  - Increased gravity by 20%
  - Increased gravity by 50%
  - Increased pole mass by 20%
  - Increased pole mass by 50%

- **Hopper-v5 Modifications**
  - Decreased floor friction by 20%
  - Decreased floor friction by 50%
  - Increased torso mass by 20%
  - Increased torso mass by 50%

- **Pusher-v5 Modifications**
  - Increased puck mass by 20%
  - Increased puck mass by 50%
  - Shifted target goal location for the puck

- **Ant-v5 Modifications**
  - Decreased joint gear ratio by 20%
  - Decreased joint gear ratio by 50%
  - Disabled one leg of the ant, altering its morphology and effective dynamics

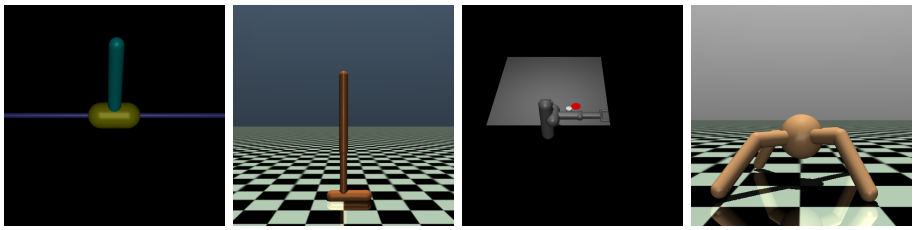

Figure 1: Environments pictured from left to right in ascending difficulty: InvertedPendulum-v5, Hopper-v5, Pusher-v5, Ant-v5.

3.3 Expert Demonstration Data

Expert demonstrations for each base environment were generated by training an SAC agent until high performance was achieved. To collect the demonstration data, actions were selected deterministically from the trained SAC expert policy by taking the mean of the policy's output distribution; no exploration noise was

added to these actions. This ensures that the demonstrations represent the learned optimal behavior of the expert.

For each base environment, a dataset corresponding to 1 million timesteps of this deterministic expert interaction was collected. Each entry in the dataset consists of (`state`, `action`, `next_state`, `ground_truth_reward`, `done`) tuples. The same demonstration dataset was used for training all imitation learning algorithms (BC, GAIL, and AIRL). Although the ground truth reward was collected, it was not used for training any of the imitation learning algorithms.

3.4 Experimental Procedure and Evaluation

Our evaluation protocol was designed to address the following research questions:

1. **Baseline Imitation Performance (RQ1):**
   - BC, GAIL, and AIRL policies are trained using the expert demonstrations from each base environment.
   - The learned policies are then evaluated in their respective base environments.
2. **Policy Robustness to Perturbations (RQ2):**
   - The policies trained in the base environments (from RQ1) are deployed *zero-shot* (i.e. without any retraining or fine-tuning) in each of the corresponding modified environments laid out in Section 3.2.
   - Performance degradation compared to the base environment is measured.
3. **AIRL Reward Quality and Transferability (RQ3):**
   - The state-only reward function $g_\phi(s)$ learned by the AIRL model from RQ1 is extracted.
   - A new SAC agent (SAC-AIRL) is trained from scratch in each of the *modified* environments using *only* this transferred AIRL reward signal. The agent does not have access to the ground-truth reward of the modified environment.
4. **Comparative Performance in Modified Environments (RQ4):**
   - The performance of the SAC-AIRL agent from RQ3 in the modified environments is compared against:
     - The zero-shot performance of the BC policy from RQ2.

  - The zero-shot performance of the GAIL policy from RQ2.
  - The zero-shot performance of the AIRL policy from RQ2.
- Additionally, as an approximate upper-bound for performance in modified environments, an Expert SAC agent is trained from scratch in each modified environment using the *ground truth* reward function.

For all evaluations, policies are run for 100 episodes on randomly generated seeds, and performance is measured by the average cumulative returns. Learning curves are also analyzed to understand training dynamics, although that isn't the main focus of this study.

3.5 Implementation Details

Both GAIL and AIRL are implemented using this [18] existing codebase. BC is implemented as a standard supervised regression model. The SAC algorithm utilizes the established Stable Baselines3 library [9]. All experiments are conducted using Python, with environments provided by Gymnasium.

4 Results

This section presents the empirical results of our comparative analysis, addressing each research question outlined previously. Performance is reported as mean cumulative return and standard deviation over 100 evaluation episodes. Any numbers in bold indicate that it's either the highest cumulative return, or it's within 5% of the highest cumulative return.

4.1 RQ1: Baseline Imitation Performance

Table 1 summarizes the performance of BC, GAIL, AIRL, and the original expert SAC in the standard, unmodified environments. We also include the performance of an SAC agent trained using the AIRL-learned reward within the base environment (SAC-AIRL) as a validation of the learned reward's quality without environmental shifts. Figure 2 shows the relative performance degradation between the expert SAC model and the SAC-AIRL model. Additional details about each environment's observation space, action space, ground truth reward function, and truncation/termination conditions can be found in Appendix B.

In the Ant and Hopper environments, the expert SAC achieved the highest returns, serving as the performance upper bound. All IL methods (AIRL, BC,

Table 1: Algorithm Performance on Standard Environments (Mean ± Std Return)

	Ant	Hopper	Inverted Pendulum	Pusher
Expert SAC	**5655.26 ± 882.56**	**4070.47 ± 307.83**	**1000.00 ± 0.00**	**-30.53 ± 7.83**
BC	5204.70 ± 1386.33	3233.93 ± 416.85	**1000.00 ± 0.00**	**-30.49 ± 11.30**
GAIL	4576.90 ± 1649.49	3476.49 ± 1.49	**1000.00 ± 0.00**	-35.51 ± 6.41
AIRL	5233.60 ± 989.55	3410.27 ± 18.06	**1000.00 ± 0.00**	-45.09 ± 5.98
SAC-AIRL	2865.51 ± 1029.50	3363.64 ± 378.97	**1000.00 ± 0.00**	-52.08 ± 5.58

GAIL) successfully learned policies that achieved substantial positive returns, though generally lower than the expert SAC. The SAC-AIRL model achieved a return of 2865.51 in Ant and 3363.64 in Hopper indicating that the AIRL reward function, even in the base environment, captures a significant portion of the task objective. However, it does not fully replicate the ground truth reward's efficacy for training an SAC agent to expert level.

For the InvertedPendulum environment, all evaluated algorithms consistently achieved the maximum possible score of 1000.00, demonstrating successful task mastery. However, this was expected given the simplicity of the environment.

For the Pusher environment, all of the imitation learning models were able to approach expert level performance. However, the SAC-AIRL model's score is misleading as it doesn't indicate whether or not the agent was able to successfully push the puck into the goal location. Viewing the agent during evaluation shows that it was unable to successfully complete the task which offers some insight into the potential limitations of AIRL's learned reward function. AIRL was able to successfully complete the task, but was penalized for extraneous actions that the other algorithms did not take. This suggests that the AIRL reward function was unable to capture the penalty for actions in the ground truth reward which makes sense given we used the state only discriminator formulation.

4.2 RQ2: Policy Robustness to Environmental Variations

We evaluated the zero-shot robustness of policies trained in the base environments when deployed in modified environments as described in Section 3.2. The following sections present the results for each environment as well as a brief list of notable takeaways. The expert SAC algorithm was trained on the ground-truth reward for each of these modifications, and the SAC-AIRL algorithm was trained on the AIRL reward. The other algorithms were deployed zero-shot.

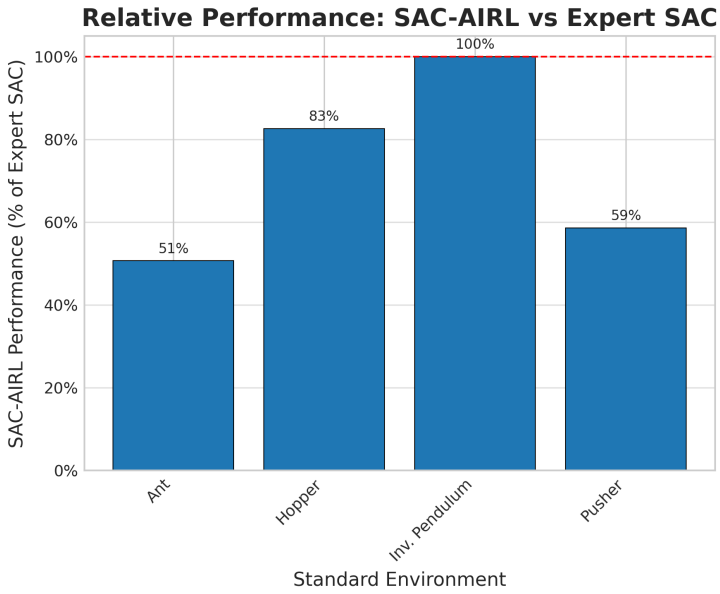

Figure 2: Relative performance degradation between the expert SAC model and the SAC-AIRL model. It is calculated as SAC-AIRL performance divided by expert SAC performance and gives a good idea of how well AIRL was able to capture the ground-truth reward.

InvertedPendulum Environment Modifications

Table 2 shows the results for InvertedPendulum modifications.

- Both AIRL and GAIL demonstrated perfect zero-shot robustness, maintaining the maximum score of 1000.00 across all tested modifications ("Gravity +20%/+50%", "Pole Mass +20%/+50%"), matching the Expert SAC oracle. The relative simplicity of the InvertedPendulum task and its clear objective likely contribute to the expert policies learned by AIRL and GAIL being inherently robust to these specific dynamic changes, as the core balancing strategy remains largely the same.
- BC also achieved a perfect score under "Gravity +20%" but saw a significant drop with "Gravity +50%" (421.98) and moderate degradation with pole mass changes (986.47 and 871.55). This again highlights BC's sensitivity to larger distributional shifts, even in simpler environments, when the agent encounters states not well-represented in the fixed demonstration dataset.

Table 2: Algorithm Performance on Modified Inverted Pendulum Environments (Mean ± Std Return)

	Gravity +20%	Gravity +50%	Pole Mass +20%	Pole Mass +50%
Expert SAC	**1000.00 ± 0.00**	**1000.00 ± 0.00**	**1000.00 ± 0.00**	**1000.00 ± 0.00**
BC	**1000.00 ± 0.00**	421.98 ± 267.17	986.47 ± 83.89	871.55 ± 251.55
GAIL	**1000.00 ± 0.00**	**1000.00 ± 0.00**	**1000.00 ± 0.00**	**1000.00 ± 0.00**
AIRL	**1000.00 ± 0.00**	**1000.00 ± 0.00**	**1000.00 ± 0.00**	**1000.00 ± 0.00**
SAC-AIRL	**1000.00 ± 0.00**	**1000.00 ± 0.00**	**1000.00 ± 0.00**	**1000.00 ± 0.00**

Hopper Environment Modifications

The robustness of policies in modified Hopper environments is detailed in Table 3.

- AIRL displayed notable zero-shot robustness in Hopper. For "Friction -20%" and "Torso Mass +20%", AIRL achieved high scores of 3292.83 and 3498.83, respectively. In these scenarios, BC and GAIL experienced significant performance drops. AIRL's potential to learn more disentangled or salient features of the expert's behavior might contribute to its better generalization under these dynamic changes compared to BC's direct state-action mapping or GAIL's distribution matching which might overfit to specifics of the original dynamics.
- Under more severe changes ("Friction -50%" and "Torso Mass +50%"), all zero-shot IL policies degraded considerably. However, AIRL (751.24 and 1810.75) generally retained more performance than BC (490.32 and 781.39) and GAIL (578.50 and 831.86), particularly in the "Torso Mass +50%" case.

Table 3: Algorithm Performance on Modified Hopper Environments (Mean ± Std Return)

	Friction -20%	Friction -50%	Torso Mass +20%	Torso Mass +50%
Expert SAC	2955.07 ± 160.87	**2918.38 ± 456.25**	2589.99 ± 439.62	**2968.82 ± 26.46**
BC	880.35 ± 25.26	490.32 ± 29.09	1413.67 ± 528.24	781.39 ± 63.29
GAIL	1107.00 ± 55.79	578.50 ± 6.61	1416.40 ± 510.31	831.86 ± 33.17
AIRL	**3292.83 ± 82.85**	751.24 ± 43.32	**3498.83 ± 3.20**	1810.75 ± 171.19
SAC-AIRL	2955.07 ± 160.87	**2929.59 ± 483.66**	2590.57 ± 435.91	**2967.84 ± 27.05**

Pusher Environment Modifications

Performance in the modified Pusher environments is given in Table 4. It is important to note that negative returns are expected in this environment due to

the nature of its ground-truth reward formulation, where penalties are incurred.

- The Pusher environment proved highly challenging for zero-shot transfer for all IL methods. All IL policies (BC, GAIL, AIRL) exhibited substantial performance degradation compared to the Expert SAC (Oracle) across all modifications ("Goal shift", "Puck Mass +20%", "Puck Mass +50%"). For instance, under the "Goal shift" condition, BC achieved -58.51, GAIL -68.40, and AIRL -85.29, all considerably worse than the Expert SAC oracle's -22.53.
- Among the zero-shot IL methods, BC generally performed slightly better (less negatively) than GAIL and AIRL, particularly under the "Goal shift" and "Puck Mass +20%" conditions. However, all IL methods struggled significantly, suggesting that the policies learned in the base environment were not robust to these specific perturbations in a task requiring precise manipulation.
- **Failure in Pusher:** Consistent with the poor zero-shot performance, training with the transferred AIRL reward in Pusher modifications (Table 4) did not yield effective policies. The SAC-AIRL agent resulted in significantly more negative returns (e.g., -80.32 for "Goal shift", -56.41 for "Puck Mass +20%") compared to both the zero-shot IL policies and, most notably, the Expert SAC (Oracle). This outcome strongly suggests that the AIRL reward function learned in the base Pusher environment failed to capture a generalizable representation of the task objective. A key reason for this failure, particularly for the "Goal shift" modification, is likely that AIRL was trained exclusively on expert trajectories demonstrating behavior towards the original, default goal location. Consequently, the learned reward function may have overfitted to features associated with that specific goal position (e.g., absolute coordinates of the puck relative to that goal or the agent's end-effector approaching that specific region), rather than learning a more abstract concept of "moving the puck to a target." When the goal was shifted, this non-generalizable reward signal would misguide the SAC-AIRL agent, leading to its poor performance. Similarly, changes in puck mass might alter dynamics in ways that the original reward features, learned under fixed mass conditions, could not properly account for.

Ant Environment Modifications

Table 5 presents the performance of algorithms on modified Ant environments.

- Under the "Disabled leg" modification, the Expert SAC (Oracle) achieved the highest performance by a large margin (4204.55). Among the zero-shot

Table 4: Algorithm Performance on Modified Pusher Environments (Mean ± Std Return)

	Goal shift	Puck Mass +20%	Puck Mass +50%
Expert SAC	**-22.53 ± 2.98**	**-22.69 ± 2.91**	**-25.74 ± 4.98**
BC	-58.51 ± 9.63	-29.00 ± 13.15	-30.75 ± 15.15
GAIL	-68.40 ± 8.98	-34.99 ± 6.83	-34.19 ± 7.83
AIRL	-85.29 ± 17.93	-45.14 ± 5.88	-44.73 ± 6.01
SAC-AIRL	-80.32 ± 14.03	-56.41 ± 6.01	-52.23 ± 5.53

IL methods, AIRL (1023.47) and GAIL (1002.13) demonstrated notable robustness by maintaining positive returns, while BC (844.20) was somewhat lower. However, all IL methods showed a significant performance gap compared to the oracle, underscoring the severity of this morphological change for policies trained on the standard four-legged Ant.

- For the "Gear -20%" modification, BC (5171.09) and AIRL (5098.03) exhibited exceptional robustness, achieving the highest scores and slightly surpassing even the strong performance of the Expert SAC (Oracle) (4997.17). GAIL (4917.77) also performed robustly, close to the oracle. We hypothesize that the slight decrease in gear ratio acts as a regularizer for the zero-shot policies since the standard deviations are lower than they are in the regular environment.

- With the more substantial "Gear -50%" change, the Expert SAC (Oracle) again set the performance benchmark (2968.44). Among the IL methods, AIRL (2793.96) was the most robust, achieving a score close to the oracle, followed by GAIL (2447.46). BC's performance (1710.13) degraded more significantly, likely reflecting its susceptibility to compounding errors under larger distributional shifts, a weakness that adversarial methods like AIRL and GAIL aim to mitigate.

Table 5: Algorithm Performance on Modified Ant Environments (Mean ± Std Return)

	Disabled leg	Gear -20%	Gear -50%
Expert SAC	**4204.55 ± 1700.08**	**4997.17 ± 1023.66**	**2968.44 ± 874.17**
BC	844.20 ± 406.90	**5171.09 ± 80.21**	1710.13 ± 191.90
GAIL	1002.13 ± 564.12	4917.77 ± 738.36	2447.46 ± 217.78
AIRL	1023.47 ± 491.89	**5098.03 ± 82.18**	2793.96 ± 173.87
SAC-AIRL	-285.85 ± 128.88	2318.00 ± 80.25	982.31 ± 2.49

4.3 RQ3 & RQ4: AIRL Reward Transferability and Comparative Performance

We investigated the quality of the AIRL-learned reward function by using it to train an SAC agent from scratch in the modified environments, denoted as SAC-AIRL. Its performance was compared against the zero-shot IL policies and the Expert SAC (Oracle). We discuss the notable outcomes, referencing the same Tables 2, 3, 4, and 5.

- **Successful Transfer in Hopper:** As seen in Table 3, reward transfer was highly effective in the Hopper environment. The SAC-AIRL agent consistently achieved performance levels very close to, or matching, the Expert SAC across all four modifications. In all Hopper modifications, SAC-AIRL significantly outperformed all zero-shot IL policies. This strong result suggests that the state-only reward learned by AIRL for Hopper captured generalizable features of the task that remained relevant even under considerable dynamics changes.
- **Perfect Transfer in InvertedPendulum:** Similarly, in InvertedPendulum (Table 2), SAC-AIRL achieved perfect scores (1000.00) in all modified environments, matching the Expert SAC. This indicates that for simpler tasks where the optimal policy is less sensitive to the tested dynamics variations, AIRL's learned reward can be effectively transferred.
- **Mixed Results in Ant:** The efficacy of reward transfer in Ant (Table 5) remained mixed, and the performance of SAC-AIRL was notably suboptimal when compared against the updated, stronger Expert SAC baselines. For the "Disabled leg" modification, SAC-AIRL performed very poorly (-285.85), starkly contrasting with the Oracle's 4204.55 and also performing worse than all zero-shot IL methods. This underscores a clear failure of the transferred reward to guide learning effectively under this drastic morphological change, likely because the state features AIRL learned as relevant for reward in the four-legged Ant were insufficient or misleading for a three-legged agent. For the gear ratio changes, SAC-AIRL achieved 2318.00 (for -20%) and 982.31 (for -50%). While these represent successful learning to some extent, these scores are considerably lower than the respective Expert SAC (Oracle) performances (4997.17 and 2968.44) and also lag behind the best zero-shot IL methods (BC/AIRL for -20% gear, and AIRL/GAIL for -50% gear). This indicates that while the transferred reward provided some useful signal, it was not sufficient to recover near-oracle performance or even match the best direct IL generalizations for these dynamic changes.
- **Failure in Pusher:** Consistent with the poor zero-shot performance, training with the transferred AIRL reward in Pusher modifications (Table

4) did not yield effective policies. The SAC-AIRL agent resulted in significantly more negative returns (e.g., -80.32 for "Goal shift", -56.41 for "Puck Mass +20%") compared to both the zero-shot IL policies and, most notably, the Expert SAC (Oracle). This outcome strongly suggests that the AIRL reward function learned in the base Pusher environment failed to capture a generalizable representation of the task objective. A key reason for this failure, particularly for the "Goal shift" modification, is likely that AIRL was trained exclusively on expert trajectories demonstrating behavior towards the original, default goal location. Consequently, the learned reward function may have overfitted to features associated with that specific goal position (e.g., absolute coordinates of the puck relative to that goal or the agent's end-effector approaching that specific region), rather than learning a more abstract concept of "moving the puck to a target." When the goal was shifted, this non-generalizable reward signal would misguide the SAC-AIRL agent, leading to its poor performance. Similarly, changes in puck mass might alter dynamics in ways that the original reward features, learned under fixed mass conditions, could not properly account for.

In summary for reward transfer, when effective (e.g., Hopper, InvertedPendulum), retraining with the AIRL reward SAC-AIRL led to substantial gains over zero-shot policies, often approaching oracle performance. However, its success was environment and modification dependent, with notable failures indicating that the generalizability of learned reward functions can be severely challenged by specific types of environmental or morphological shifts not represented in the demonstration data.

5 Future Work

This research has provided a comprehensive empirical comparison of IL algorithms, focusing on robustness and reward transferability. The findings open up several promising avenues for future investigation.

1. **Improving Reward Generalization under Severe Shifts:**
    - The observed failures in reward transfer for SAC-AIRL under significant perturbations (e.g., Pusher goal shifts, Ant disabled leg) highlight a critical challenge. Future work could explore methods to learn more invariant or abstract reward functions. This might involve:
        - *Data Augmentation for Demonstrations:* Systematically augmenting expert demonstration data with variations in dynamics or task parameters before training AIRL could expose the IRL

algorithm to a wider range of conditions, potentially leading to more robust reward features [8].
   - *Adversarial Training for Reward Robustness:* Incorporating techniques inspired by adversarial training from supervised learning. This would involve explicitly challenging the learned reward function during its training by trying to find small changes to the state that maximally alter the predicted reward, and then training the reward functino to be less sensitive to such changes. This could encourage the reward to focus on more inherently robust and task-relevant features.

2. **Enhancing Zero-Shot Policy Robustness:** While AIRL showed better zero-shot robustness in several cases, further improvements are desirable. Future research could investigate:
   - *Domain Randomization during IL Policy Training:* Applying domain randomization during the policy optimization phase of GAIL or AIRL could lead to policies that are inherently more robust to dynamics variations [15].
   - *Ensemble Methods for IL Policies:* Training an ensemble of IL policies and using an aggregation strategy at test time might improve overall robustness [2].

3. **Quantifying and Predicting Transferability:** It would be valuable to develop metrics or methods that can predict *a priori* how well a learned reward function (like AIRL's) or an IL policy will transfer to a new, modified environment. This could save significant computational effort in retraining and allow for more informed decisions about when transfer is likely to be successful. This might involve analyzing the feature representations learned by the discriminator or policy.

4. **Broader Range of Environments and Modifications:** Expanding the empirical study to a wider array of environments (e.g., with visual inputs, more complex contact dynamics, or partially observable states) and a more diverse set of modifications (e.g., sensory noise, actuator failures, different opponent strategies in multi-agent settings) would further test the limits of current IL and IRL techniques.

5. **Combining Strengths of Different IL/IRL Approaches:** Investigating hybrid approaches that combine the strengths of different methods. For example, could the explicit reward from AIRL be used to guide or regularize the policy learning in GAIL, or could BC be used to pre-train policies for faster convergence in adversarial IL setups, especially in the context of transfer?

6. **Theoretical Understanding of Robustness and Transfer:** Further theoretical work is needed to better understand the conditions under which

IL/IRL methods can guarantee robustness or successful reward/policy transfer, particularly for adversarial approaches like GAIL and AIRL.

6 Conclusions

This paper presented a comprehensive empirical investigation into the baseline performance, zero-shot robustness, and reward transferability of several prominent imitation learning (IL) and inverse reinforcement learning (IRL) algorithms: Behavioral Cloning (BC), Generative Adversarial Imitation Learning (GAIL), and Adversarial Inverse Reinforcement Learning (AIRL). Our experiments, conducted across multiple MuJoCo environments with various dynamics and task modifications, yielded several key findings.

Firstly, regarding baseline imitation (RQ1), all IL methods successfully learned competent policies in the original environments, though a performance gap to the expert often remained, particularly in more complex tasks like 'Ant' and 'Hopper'. Training an SAC agent with the AIRL-learned reward in the base environment (`SAC-AIRL (Transfer)`) demonstrated that AIRL could capture a significant portion of the task objective, though typically not to the full extent of the ground-truth reward.

Secondly, in terms of zero-shot policy robustness to environmental perturbations (RQ2), AIRL generally exhibited superior generalization compared to BC and GAIL, especially in 'Ant' and 'Hopper' under moderate dynamics changes. However, all zero-shot policies, including AIRL's, struggled significantly when faced with substantial shifts or in environments like 'Pusher' that are highly sensitive to parameter changes, underscoring the inherent limitations of deploying IL policies directly in out-of-distribution settings. BC, as expected, was often the most susceptible to performance degradation due to covariate shift.

Thirdly, investigating AIRL's reward transferability (RQ3), we found that the quality of the transferred reward was highly context-dependent. In environments like 'Hopper' and 'InvertedPendulum', the AIRL-learned reward successfully guided a new SAC agent (`SAC-AIRL (Transfer)`) to achieve performance levels comparable to an oracle trained with ground-truth rewards in the modified settings. This demonstrated the potential of AIRL to learn generalizable reward functions. However, for other environments ('Pusher') or specific types of modifications (e.g., morphological changes like the 'Ant' disabled leg, or 'Pusher' goal shifts where demonstrations were limited to a single goal), the transferred reward failed to facilitate effective learning, sometimes performing worse than zero-shot policies. This highlighted that reward functions learned from demonstrations in a limited context may not capture sufficiently abstract or robust task representations for broad generalization.

Finally, comparing the transferred reward approach to zero-shot policies (RQ4), our results showed that when AIRL's reward transfer was successful, retraining an agent with this reward significantly outperformed all zero-shot IL policies in the modified environments. This underscores the value of leveraging learned rewards for adaptation when the reward signal itself generalizes adequately.

In conclusion, while advanced IRL methods like AIRL offer promising avenues for learning transferable reward functions and improving policy robustness over simpler IL techniques, significant challenges remain in achieving consistent generalization across diverse environmental shifts and task alterations. The success of reward transfer appears contingent on the nature of the task, the type of modification, and the diversity of the experiences from which the reward was learned. Future work, as outlined in Section 5, should prioritize strategies for enhancing reward generalization and policy robustness, alongside developing a deeper theoretical understanding and broader empirical validation of these imitation learning approaches.

References

[1] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in ai safety, 2016.

[2] T. G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems*, pages 1–15, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[3] J. Fu, K. Luo, and S. Levine. Learning robust rewards with adversarial inverse reinforcement learning. *CoRR*, abs/1710.11248, 2017.

[4] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014.

[5] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[6] J. Ho and S. Ermon. Generative adversarial imitation learning, 2016.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015.

[8] L. Perez and J. Wang. The effectiveness of data augmentation in image classification using deep learning, 2017.

[9] 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.

[10] S. Ross, G. J. Gordon, and J. A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning, 2011.

[11] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016.

[12] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization, 2017.

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.

[14] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[15] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world, 2017.

[16] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033. IEEE, 2012.

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

[18] T. Watanabe. *toshikwa/gail-airl-ppo.pytorch*. 11 2020.

A Implementation Details

A.1 General Setup

All experiments were conducted using Python 3.11, PyTorch 2.7.0, Gymnasium 1.1.1, and Stable Baselines3 2.6.0. Training was performed on a single NVIDIA A100 GPU. Evaluation results reported in the main paper are averages over 100 random seeds, with standard deviations shown.

A.2 Algorithm Configurations

The general network architectures for policy and value functions across algorithms consisted of Multi-Layer Perceptrons (MLPs) with two hidden layers, typically using ReLU activation functions. Specific layer sizes and minor architectural variations are detailed in the configuration files.

Soft Actor-Critic (SAC) was used for expert demonstration generation and for the SAC-AIRL agent. Behavioral Cloning (BC) was implemented as a supervised regression model. GAIL and AIRL utilized PPO for their policy optimization.

Key hyperparameters, such as learning rates, batch sizes, and discount factors, were tuned for each algorithm and environment combination. Total training timesteps varied based on environment complexity, for example, 2×10^7 steps were used for AIRL in Ant-v5, while 2.5×10^5 steps were used for InvertedPendulum-v5.

For a complete and exhaustive list of all hyperparameters used for each algorithm in each environment, please refer to the YAML configuration files provided in our supplementary code repository: `https://github.com/nvan21/Honors-Capstone-IL-Robustness/tree/main`. The repository also contains the code used to run all experiments.

B Environment Details

This appendix provides details on the MuJoCo environments used in this study, sourced from the Gymnasium MuJoCo suite [**?**]. For complete and official specifications, including precise observation and action space details, readers are referred to the Gymnasium online documentation (`https://gymnasium.farama.org/environments/mujoco/`).

The modified XML model files defining the altered physics or morphology for each environment are available in our repository within the `xml/` directory.

B.1 InvertedPendulum-v5

Figure 3 provides a visual representation of the InvertedPendulum environment.

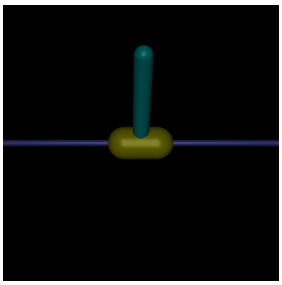

Figure 3: The InvertedPendulum-v5 environment.

Standard Environment

- **Objective:** A cart-pole system where the goal is to balance a pole upright on a cart that moves along a track by applying forces to the cart.
- **State Space (Observation Space):** A vector of 4 continuous values: position of the cart along the surface, vertical angle of the pole, linear velocity of the cart, and pole angular velocity at the tip.
- **Action Space:** A single continuous value representing the force applied to the cart, bounded between [-3, 3] and then scaled to [-1, 1].
- **Ground-Truth Reward Function:** The reward r_t at each step t is:

$$r_t = 1.0 \tag{7}$$

  A reward of +1.0 is given for every timestep the pole remains upright (e.g., |pole angle| < 0.2 radians) and the cart remains on track. Termination occurs if these conditions are violated or the episode reaches a maximum length of 1000 steps.

Modifications Implemented

- **Gravity +20%:** The gravitational acceleration constant in the simulation's XML model was increased by 20%.

- **Gravity +50%:** Gravitational acceleration was increased by 50%.
- **Pole Mass +20%:** The mass of the pole component in the XML model was increased by 20%.
- **Pole Mass +50%:** The mass of the pole component was increased by 50%.

B.2 Hopper-v5

Figure 4 provides a visual representation of the Hopper environment.

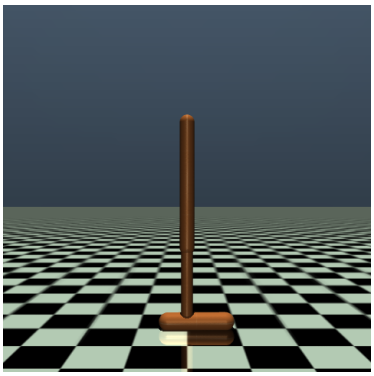

Figure 4: The Hopper-v5 environment.

Standard Environment

- **Objective:** The Hopper is a 2D one-legged robot. The goal is to make it hop forward (positive x-direction) as fast as possible without falling over.
- **State Space (Observation Space):** A vector of 11 continuous values. These include the height of the torso, its angle, angular velocities of joints, and linear velocities of the torso.
- **Action Space:** A vector of 3 continuous values representing the torques applied to the thigh, leg, and foot joints, bounded between [-1, 1].
- **Ground-Truth Reward Function:** The reward r_t at each step t is defined as:

$$r_t = v_x + r_{\text{healthy}} - w_{\text{ctrl}} \sum_i a_i^2 \tag{8}$$

where v_x is the forward velocity of the torso, r_{healthy} is a constant positive bonus (e.g., +1) awarded if the agent's state is within healthy operational

limits (e.g., torso height $z \in [0.7, \infty)$, torso angle $|\text{angle}| < 0.2$), and w_{ctrl} is a small positive weight (e.g., 10^{-3}) for the control cost penalty based on actions a_i. Termination occurs if the agent falls or the episode reaches 1000 steps.

Modifications Implemented

- **Friction -20%:** The friction coefficient of the Hopper's feet were reduced by 20% from its original value.
- **Friction -50%:** Friction coefficients were reduced by 50%.
- **Torso Mass +20%:** The mass of the Hopper's torso body component was increased by 20%.
- **Torso Mass +50%:** The mass of the Hopper's torso was increased by 50%.

B.3 Pusher-v5

Figure 5 provides a visual representation of the Pusher environment.

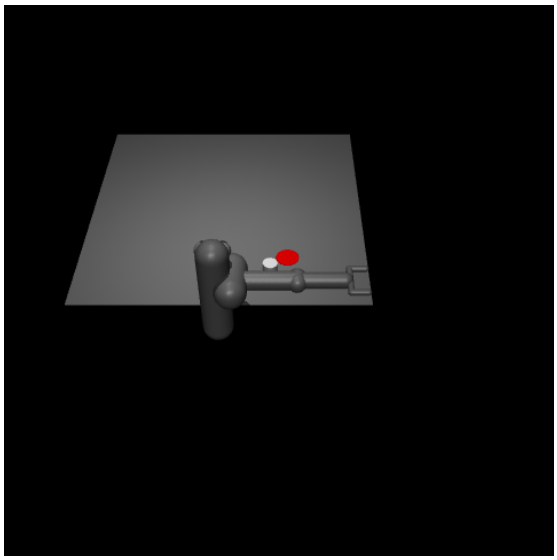

Figure 5: The Pusher-v5 environment.

Standard Environment

- **Objective:** A 7-DoF robotic arm (the "pusher") must move a puck (cylinder) on a table to a target goal location using its end-effector.
- **State Space (Observation Space):** A vector of 23 continuous values, including joint angles and velocities of the arm, end-effector position, and the position and velocity of the puck.
- **Action Space:** A vector of 7 continuous values representing torques applied to the arm's joints, bounded between [-2, 2] and then scaled to [-1, 1].
- **Ground-Truth Reward Function:** The reward r_t at each step t is structured to encourage moving the puck towards the goal and the arm's end-effector towards the puck, resulting in negative values where less negative is better:

$$r_t = -w_{\text{tip\_puck}} d(\text{fingertip}, \text{puck}) - w_{\text{puck\_goal}} d(\text{puck}, \text{goal}) - w_{\text{ctrl}} \sum_i a_i^2 \tag{9}$$

where $d(\cdot, \cdot)$ is the Euclidean distance, a_i are the actions, and $w_{\text{tip\_puck}}$, $w_{\text{puck\_goal}}$, and w_{ctrl} are positive weighting coefficients. The default weights are $w_{\text{tip\_puck}} = 0.5$, $w_{\text{puck\_goal}} = 1$, $w_{\text{ctrl}} = 0.1$. The episode runs for 100 steps.

Modifications Implemented

- **Goal Shift:** The target (x, y) coordinates for the puck on the table were shifted from their default location to a new location.
- **Puck Mass +20%:** The mass of the puck (cylinder object) in the XML model was increased by 20%.
- **Puck Mass +50%:** The mass of the puck was increased by 50%.

B.4 Ant-v5

Figure 6 provides a visual representation of the Ant environment.

Standard Environment

- **Objective:** The Ant is a 3D quadrupedal robot. The goal is to make it move forward (positive x-direction) as fast as possible without flipping over.

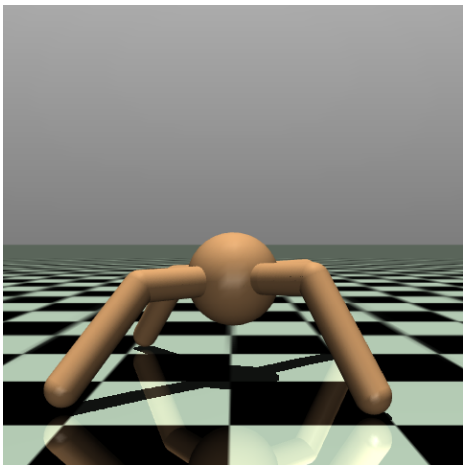

Figure 6: The Ant-v5 environment.

- **State Space (Observation Space):** A vector of 27 continuous values. These include positions (excluding x,y of torso), orientations (e.g., torso quaternion), joint angles, velocities, and contact forces.
- **Action Space:** A vector of 8 continuous values representing torques applied to the hip and ankle joints of its four legs, bounded between [-1, 1].
- **Ground-Truth Reward Function:** The total reward r_t at each step t is defined as:

$$r_t = r_{\text{healthy}} + r_{\text{forward}} - r_{\text{ctrl\_cost}} - r_{\text{contact\_cost}} \tag{10}$$

  where the components are:

  - r_{healthy}: A fixed reward (default is +1.0) awarded at every timestep the Ant is considered "healthy." The definition of healthy is based on state variable limits (e.g., torso height $z \in [0.2, 1.0]$ and valid contact configurations, see Gymnasium documentation for precise conditions).
  - r_{forward}: Reward for forward movement. This is calculated as $w_{\text{forward}} \times \frac{dx}{dt}$, where dx is the displacement of the Ant's main body (torso) in the positive x-direction between the previous state ($x_{\text{before-action}}$) and the current state ($x_{\text{after-action}}$). dt is the time elapsed between actions, which is 'frame_skip' (default 5) multiplied by 'frametime' (default 0.01s), resulting in a default $dt = 0.05$s. The weight w_{forward} is 'forward_reward_weight' (default 1.0).
  - $r_{\text{ctrl\_cost}}$: A penalty for taking large actions, calculated as $w_{\text{control}} \times \|\mathbf{a}\|_2^2$, where $\mathbf{a}$ is the action vector and w_{control} is the 'ctrl_cost_weight' (default 0.5). This term is subtracted.

– $r_{\text{contact\_cost}}$: A penalty for excessive external contact forces, calculated as $w_{\text{contact}} \times \|\mathbf{F}_{\text{contact}}\|_2^2$. Here, $\mathbf{F}_{\text{contact}}$ represents the external contact forces (e.g., between non-foot body parts and the ground), often clipped by a 'contact_force_range'. The weight w_{contact} is the 'contact_cost_weight' (default 5×10^{-4}). This term is subtracted.

Termination occurs if the Ant is no longer "healthy" (e.g., flips over by $z < 0.2$ or $z > 1.0$) or if the maximum episode length (1000 steps) is reached.

Modifications Implemented

- **Gear -20%:** The gear ratios of all actuators (joints) were reduced by 20% from their default values in the XML model.
- **Gear -50%:** Gear ratios were reduced by 50%.
- **Disabled Leg:** One of the Ant's four legs was rendered non-functional by setting its corresponding actuator gear values in the XML model to effectively zero, preventing it from exerting controlled torque.