(a) Navigation (b) Locomotion (c) Manipulation

Figure 7: Common data augmentation functions. (7a, Amazon warehouse robots (2008)): If a robot is moving in free space, transition dynamics are often invariant to the agent's position. (7b, OSU's Cassie robot (2022)) Since robots are often symmetric about their sagittal axis, we can reflect the robot's left and right movements. (7c, Fetch robot (2023)) Objects move only if the agent contacts it. Thus, if the agent and object are not in contact, their dynamics are independent.

A DYNAMICS-INVARIANT DATA AUGMENTATION FUNCTIONS

In this section, we further motivate our focus on dynamics-invariant data augmentation functions. Specifying a dynamics-invariant data augmentation function requires knowledge of domain-specific invariances or symmetries. While domain knowledge may seem like a limitation, we observe in the literature and real-world RL applications that such invariances and symmetries are incredibly common and often require very little prior knowledge to specify. We provided a few examples:

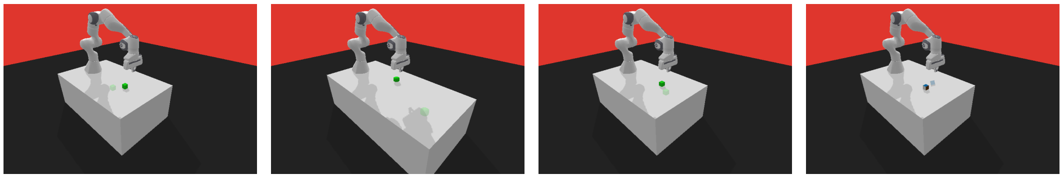
1. Transition dynamics are often independent of the agent's goal state (Andrychowicz et al., 2017).
2. Objects often have independent dynamics if they are physically separated (Pitis et al., 2020; 2022), which implies that objects exhibit translational invariance conditioned on physical separation.
3. Several works focus on rotational symmetry of 3D scenes in robotics tasks (Wang et al., 2022; 2023), and many real-world robots are symmetric in design and thus have symmetries in their transition dynamics (Mikhail Pavlov and Plis, 2018; Abdolhosseini et al., 2019).

We include real-world tasks that exhibit one or more of these invariances in Fig. 7. We choose to focus on dynamics-invariant data augmentations because they have already appeared so widely in the literature. As RL becomes an increasingly widely used tool, we anticipate that domain experts will be able to identify new domain-specific augmentations and use them to further lower the data requirements of RL. These observations underscore the importance of identifying when and why different general properties of data augmentation will benefit RL.

B PRIMARY ENVIRONMENTS FOR OUR EMPRICAL ANALYSIS

We use four tasks from `panda-gym` (Gallouédec et al., 2021) as the core environments in the main paper. Fig. 8 shows renderings for each task.

- **PandaPush-v3 (Push):** The robot must push an object to a goal location on the table. The goal and initial object positions are sampled uniformly at random from $(x, y) \in [-0.15, 0.15]^2, z = 0.02$.
- **PandaSlide-v3 (Slide):** The robot must slide a puck to a goal location on the table. The initial object position is sampled uniformly at random from $[-0.15, 0.15]^2$, while the goal (x, y, z) is sampled from $x \in [0.25, 0.55], y \in [-0.15, 0.15], z = 0.015$.
- **PandaPickAndPlace-v3 (PickAndPlace):** The robot must pick up an object and move it to a goal location. With probability 0.3, the goal is on the table ($z = 0.02$), and with probability 0.7, the goal is in the air ($z \in (0.02, 0.2]$).



(a) Push (b) Slide (c) PickAndPlace (d) Flip

Figure 8: Renderings of various Panda tasks. In PandaPush-v3 and Slide, the agent must move an object to a goal position. In PandaFlip-v3, the agent must rotate an object to a goal orientation.

Episode length	at most 50 timesteps
Evaluation frequency	10,000 timesteps
Number of evaluation episodes	80
Number of environment interactions	600K (Push), 1M (Slide, Flip), 1.5M (PickAndPlace)
Random action probability	0.3
Gaussian action noise scale	0.2
# of random actions before learning	1000
Observed replay buffer size (default)	$1 \cdot 10^6$
Augmented replay buffer size (default)	$1 \cdot 10^6$
Batch size (default)	256 (Push, Slide), 512 (Flip, PickAndPlace)
Update frequency	Every 2 timesteps (observed replay ratio of 0.5)
Network	Multi-layer perceptron with hidden layers (256, 256, 256)
Optimizer	Adam (Kingma and Ba, 2014)
Learning rate	0.001
Polyak averaging coefficient (τ)	0.95

Table 1: Default hyperparameters used in all Panda tasks.

- **PandaFlip-v3 (Flip):** The robot must pick up an object and rotate it to a goal orientation. The initial object position is sampled uniformly at random from $[-0.15, 0.15]^2$, while the goal is a uniformly sampled orientation expressed a quaternion.

In Push, PickAndPlace, and Flip, the object is a cube with side length 0.04. In Slide, the object is a cylindrical puck with height 0.03 and radius 0.03. The object's z coordinate measures the distance between the center of the object and the table (*e.g.*, In Push, Slide, and PickAndPlace, $z = 0.02$ means the object is on the table).

Push, Slide, and PickAndPlace share a similar sparse reward structure. The agent receives a reward of 0 if the object is within 0.05 units of the goal and a reward of -1 otherwise. In Flip, the agent receives a reward of 0 if the object's orientation $\boldsymbol{q}$ is within 0.2 units from the goal orientation $\boldsymbol{q}_g$ under the following angle distance metric:

$$d(\boldsymbol{q}, \boldsymbol{q}_g) = 1 - (\boldsymbol{q} \cdot \boldsymbol{q}_g)^2 = \frac{1 - \cos(\theta)}{2}$$

where θ is the angle of rotation required to rotate $\boldsymbol{q}$ to $\boldsymbol{q}_g$. Otherwise, the agent receives a reward of -1.

In the toy 2D navigation task **Goal2D**, an agent must reach a fixed goal within 100 timesteps. The agent's state (x, y, x_g, y_g) contains the coordinates of agent's positions (x, y) and the goal's position (x_g, y_g). At each timestep, the agent chooses an action (r, θ) and transitions to a new position:

$$\begin{aligned} x_{t+1} &= x_t + 0.05r\cos(\theta) \\ y_{t+1} &= y_t + 0.05r\sin(\theta) \end{aligned} \tag{1}$$

Thus, the agent moves at most 0.05 units in any direction. The goal position is fixed throughout an episode. The agent receives reward $+1$ when it is within 0.05 units of the goal and reward -0.1 otherwise. Agent and goal positions are initialized uniformly at random in $[-1, +1]^2$.

C AUGMENTATION FUNCTIONS

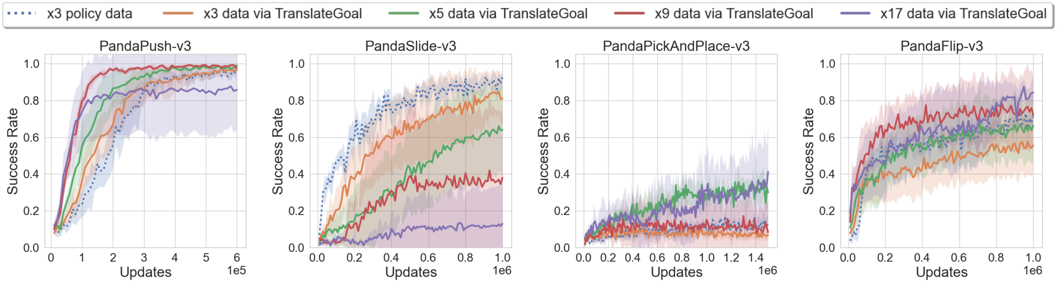
In this section, we provide further details on the data augmentation functions introduced in Section 5.1.

- TRANSLATEGOAL: Goals are relabeled using a new goal sampled uniformly at random from the goal distribution. Reward signal is generated when the new goal is sufficiently close to the object's current position. To approximate the probability of this augmentation generating reward signal in each task, we sample 10M object and goal positions uniformly at random and report the empirical probability of the goal being sufficiently close to the object to generate reward signal.
  - **PandaPush-v3 (Push):** Reward signal is generated with probability approximately 0.075.
  - **PandaSlide-v3 (Slide):** The probability of generating reward signal depends on the current policy. The initial object and goal distributions are disjoint, so this augmentation can only generate reward signal if the agent pushes the object into the region $x \in [0.25, 0.55], y \in [-0.15, 0.15]$. If the object is in this region, this augmentation will generate reward signal with probability approximately 0.075.
  - **PandaPickAndPlace-v3 (PickAndPlace):** Reward signal is generated with probability approximately 0.04.
  - **PandaFlip-v3 (Flip):** Reward signal is generated with probability approximately 0.04.
- TRANSLATEGOALPROXIMAL(p): Goals are relabeled using a new goal sampled from the goal distribution. With probability p, the new goal generates a reward signal, and with probability $1 - p$, no reward signal is generated. When generating an augmented sample with reward signal, the goal is set equal to the object's position plus a small amount of noise. and with probability $1 - p$, the object is moved to a random location sufficiently far from the goal that no reward signal is generated.

All Panda augmentation functions relabel the goal and reward. For Goal2D, we consider three data augmentation functions:

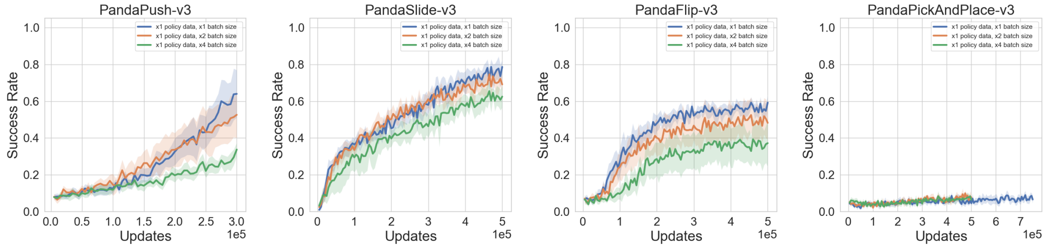
1. TRANSLATE: Translate the agent to a random position in $[-1, +1]^2$. This augmentation generates reward signal with probability approximately 0.019. We obtained this approximation by sampling 10M agent and goal positions uniformly at random and then computing the empirical probability of the goal being with 0.05 units of the agent.
2. ROTATE: Rotate both the agent and goal by $\theta \in \{\pi/2, \pi, 3\pi/2\}$. When sampling multiple augmentations of the sample observed transition, it is possible to sample duplicate augmentations.
3. TRANSLATEPROXIMAL(p): Translate the agent to a random position in $[-1, +1]^2$. With probability p, agent's new position is within 0.05 units of the goal and generates reward signal, and with probability $1 - p$, agent's new position is more than 0.05 units from the goal and generates no reward signal.

All Goal2d augmentations modify the agent's state. TRANSLATE and TRANSLATEPROXIMAL(p) modify the agent's position and reward but do not modify the goal. ROTATE affects the agent's position, the goal position, and agent's action, but does not change the reward.



(a) Push (b) Slide (c) PickAndPlace (d) Flip

Figure 9: Increasing the update ratio while keeping the augmented replay ratio fixed at $\beta = 1$ may harm performance. We plot the mean over 10 seeds expect for agents that use very large batch sizes: "x9 data" and "x17 data" curves show 5 seeds; "x17 data" for PickAndPlace shows 3 seeds. Shaded regions are 95% confidence belts. These agents are trained using the same hyperparameters as those used in Fig. 6, though the hyperparameters are slightly different than those used in other figures. See Appendix G for further details.



(a) Push (b) Slide (c) PickAndPlace (d) Flip

Figure 10: Increasing the batch size without increasing the amount of learning data available to the agent harms performance. We train agents using the hyperparameters listed in Table 1 with various batch sizes, *e.g.*, "x2 batch size" corresponds to learning with a batch size twice as large the batch size listed in Table 1.

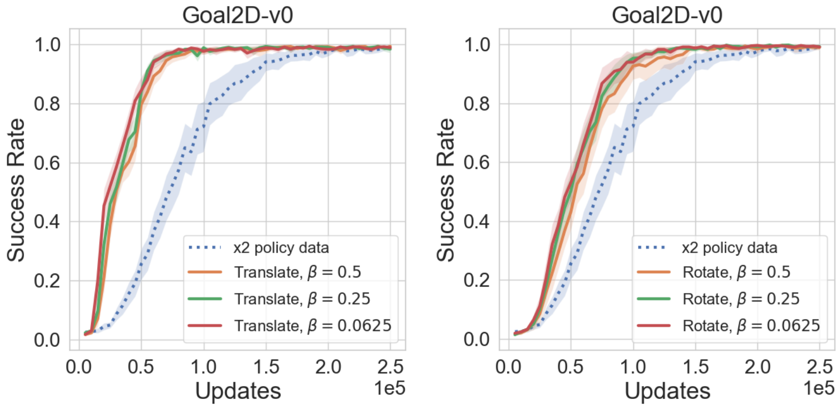
D ADDITIONAL EXPERIMENTS

D.1 INCREASING THE UPDATE RATIO

In Section 5.1.4, we demonstrate that generating more augmented data to decrease the augmented replay ratio can drastically improve data efficiency. If we generated more augmented data by increasing the augmentation ratio, we could alternatively incorporate the additional augmented data by using more augmented data in each update (*i.e.*, increasing the update ratio). Fig. 9 shows agent performance as the augmentation ratio and update ratio increase proportionally. We additionally keep the replay age fixed by increasing the augmented buffer size proportionally. Learning is generally more data efficient with a larger update ratio, though it may harm performance as seen in Slide. Notably, the improvements in data efficiency from decreasing the replay ratio (Fig. 6) are similar or better than those produced from an increased update ratio and can be achieved at a much lower computational cost per update.

D.2 INCREASING THE BATCH SIZE

In Fig. 3, agents with more training data available to them use larger batch sizes for updates, giving these agents a seemingly unfair advantage over agents that learn from less data. However, Fig. 10 shows that increasing the batch size without increasing the amount of data available to the agent harms performance, due to an increase in the expected number of times a transition is sampled for a gradient update (Fedus et al., 2020). By scaling the batch size with the amount of available learning



(a) Goal2D: TRANSLATE

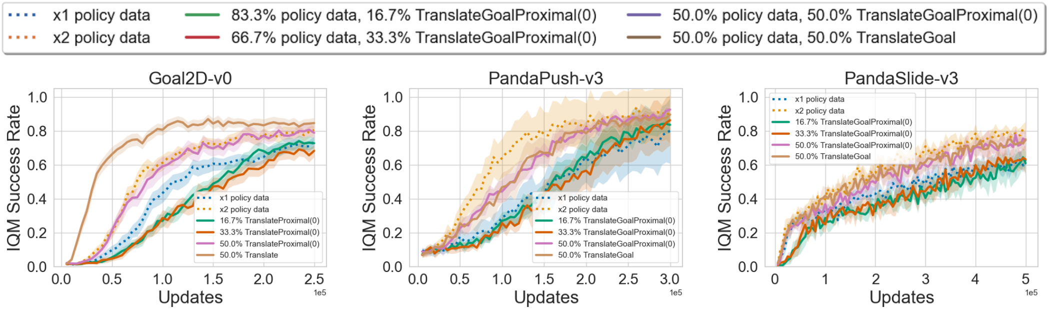
(b) Goal2D: ROTATE

Figure 11: (50 seeds) Decreasing the replay ratio while keeping the update ratio fixed at $\alpha = 1$ improves data efficiency.

data in Fig. 3, we keep the expected number of gradient updates per observed/augmented transition fixed across all experiments, providing a fairer comparison.

D.3 GOAL2D AUGMENTED REPLAY RATIO

As in Section 5.1.4, we decrease the augmented replay ratio β by generating more augmentations per observed transition. We scale the augmented replay buffer size proportionally and keep the ratio of augmented to observed data used in updates fixed at $\alpha = 1$. As shown in Fig. 11, a lower augmentation replay ratio increases data efficiency.



(a) (b) (c)

Figure 12: Learning with various mixtures of additional policy-generated data and TRANSLATEPROXIMAL(0) or TRANSLATEGOALPROXIMAL(0) data. Each mixture doubles the agent's learning data. Solid lines denote averages over 10 seeds in Panda tasks and 50 seeds in Goal2D, and shaded regions denote 95% confidence intervals. The upper legend refers to Panda tasks results. We use an update ratio of $\alpha = 1$. Panda tasks use the same hyperapameters listed in Table 1.

E GENERALIZATION EXPERIMENTS

In this section, we investigate how state-action coverage, reward density, and the augmentated replay ratio affect an agent's generalization ability. For Push, Slide, and PickAndPlace, we train agents over one quadrant of the goal distribution and evaluate agents over the full distribution. For Flip, An agent that generalizes well will achieve a high success rate over the full goal distribution. In general, our observations regarding data efficiency in the main body of this work also apply to generalization.

E.1 STATE-ACTION COVERAGE

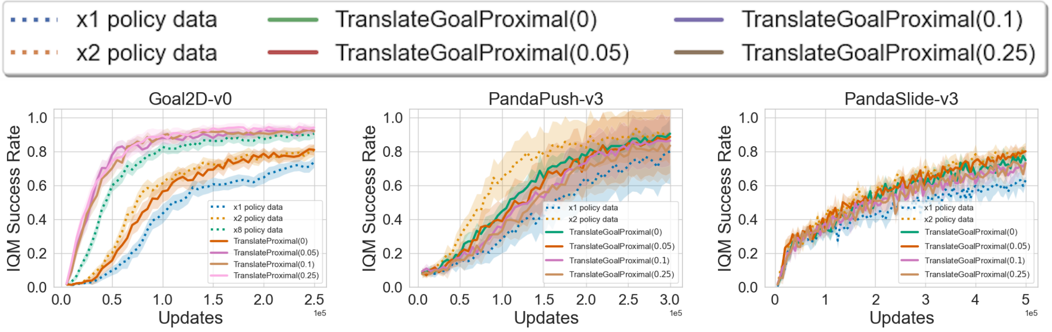
State-action coverage results are shown in Fig. 12. An increase in state-action coverage via augmentation increases generalization. In the Panda tasks, using 50% TRANSLATEGOALPROXIMAL(0) data yields similar performance compared to increased using a 50% TRANSLATEGOAL data, indicating that coverage alone can largely explain the generalization improvements with TRANSLATEGOAL. In Goal2D, increased coverage yields better generalization, though a considerable gap nevertheless exists between TRANSLATEPROXIMAL(0) and TRANSLATE. Thus, reward density must play a larger role in Goal2D. We further investigate this point in the following section.

E.2 REWARD DENSITY

Reward density results are shown in Fig. 13. In Goal2D, a relatively small increase in reward density dramatically improves generalization; TRANSLATEPROXIMAL(0) is roughly on-par with using twice as much policy-generated data, while TRANSLATEPROXIMAL(0.05) outperforms agents with x8 as much policy-generated data. In the Panda tasks, increasing reward density has little effect on data generalization. Just

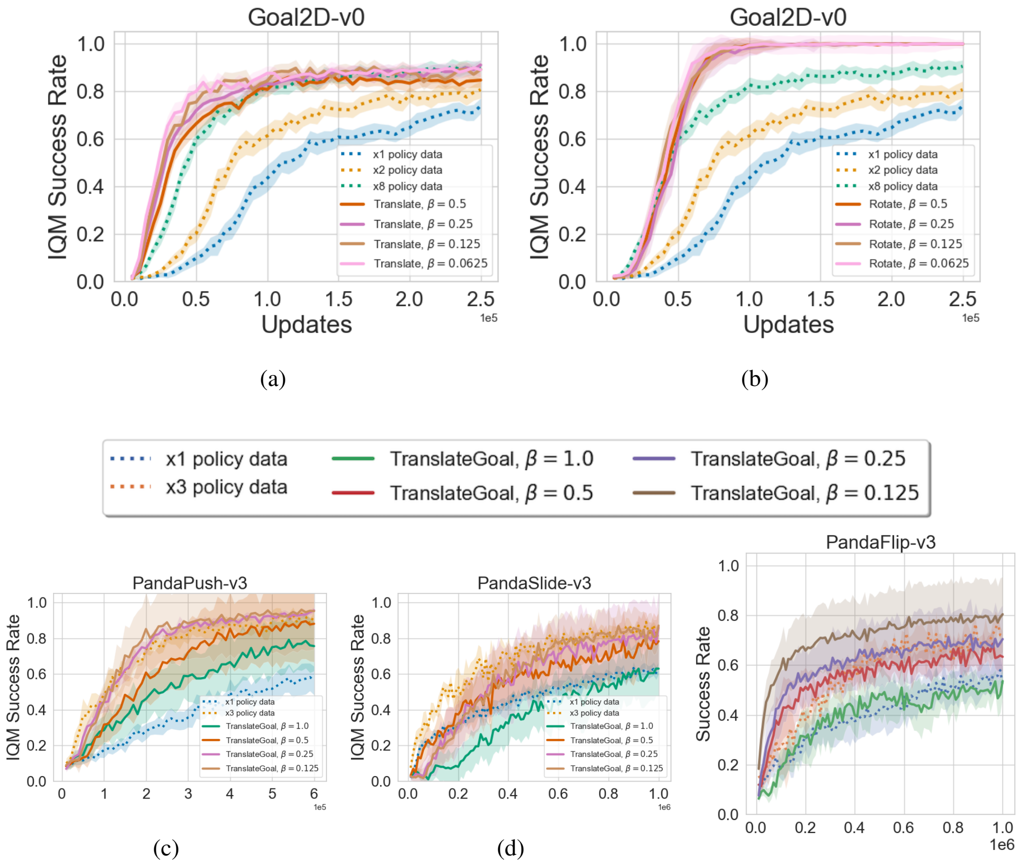
E.3 AUGMENTED REPLAY RATIO

Augmented replay ratio results are shown in Fig. 14. In Goal2D with TRANSLATE and both Panda tasks with TRANSLATEGOAL, reducing the augmented replay ratio β improves generalization performance at convergence. ROTATE achieves 100% success for all values of β.



(a) (b) (c)

Figure 13: Learning with TRANSLATEGOALPROXIMAL(p) and TRANSLATEPROXIMAL(p) for various settings of p. Solid lines denote averages over 10 seeds in Panda tasks and 50 seeds in Goal2D, and shaded regions denote 95% confidence intervals. We use an update ratio of $\alpha = 1$. Panda tasks use the same hyperapameters listed in Table 1.



(a) (b)

(c) (d)

(e)

Figure 14: Decreasing the augmented replay ratio with various augmentations. Solid lines denote averages over 10 seeds in Panda tasks and 50 seeds in Goal2D, and shaded regions denote 95% confidence intervals. We use an update ratio of $\alpha = 1$ for Goal2D and $\alpha = 2$ for Panda tasks. Panda tasks use the same hyperapameters used to in Fig. 6.

Environment	Modifications
InvertedPendulum, InvertedDoublePendulum	`cpole` `from_to` changed from $(0, 0, 0, 0.001, 0, 0.6)$ to $(0, 0, 0, 0, 0, 0.6)$.
Walker2d	`foot_left_geom` friction changed from 1.9 to 0.9 so that it matches the `foot_right_geom` friction.
Humanoid	`right_hip_y` armature changed from 0.0080 to 0.010 so that it matches `left_hip_y` armature. Specify `right_knee` stiffness of 1 so that it matches `left_knee` stiffness. Change `solver` from PGS to Newton.

Table 2: Modifications to MuJoCo environments to ensure intuitive symmetries. We include modifications to InvertedPendulum tasks even though we do not consider them in our experiments. We leave them here as a reference for future work in data augmentation that may require symmetric InvertedPendulum dynamics.

F MUJOCO EXPERIMENTS

In this appendix, we include additional experiments on the following dense reward, continuous state and action MuJoCo environments: **Swimmer-v4, Walker2d-v4, Ant-v4,** and **Humanoid-v4**. These experiments focus on state-action coverage and the augmented replay ratio, since reward density is more relevant to sparse reward tasks. With dense reward tasks, we may need to consider the full distribution of rewards in the replay buffer, not just the average.

F.1 ENVIRONMENT MODIFICATIONS

Some of the common MuJoCo environments do not exhibit symmetries that should exist intuitively (*e.g.* reflection symmetry, gait symmetry, etc.). We found two causes:

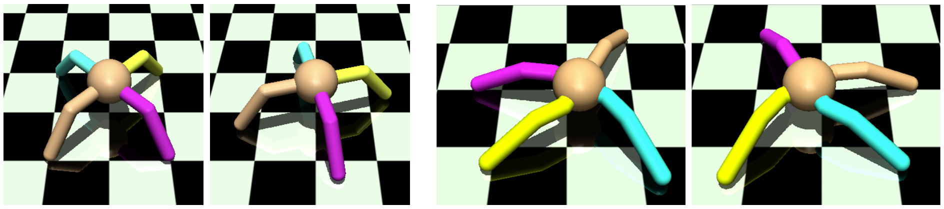
1. Asymmetric physics are explicitly hard-coded in the robot descriptor files. We believe these to be typos, and simply update values such that intuitive symmetries exist.
2. Symmetry-breaking numerical optimization algorithms are sometimes used to compute constraint forces and constrained accelerations. In particular, some environments use the Projected Gauss-Seidel (PGS) algorithm which performs sequential updates and therefore breaks symmetries where physics should be symmetric. To address this issue, we use Newton's method, which performs parallel updates and therefore preserves symmetric physics. We note that while Newton's method is MuJoCo's default algorithm, some robot descriptor files originally specify the use of PGS.

Table 2 describes all modifications made to environments to ensure intuitive symmetry. To simplify the creation of augmentation functions, we additionally exclude constraint forces and center-of-mass quantities from the agent's observations, since their interpretations are not well-documented and difficult to ascertain.

F.2 AUGMENTATION FUNCTIONS

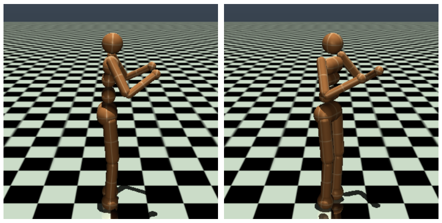
We consider the following dynamics-invariant augmentations:

- Swimmer-v4
  - REFLECT: Reflect joint angles and velocities about the agent's central axis. The reward is unchanged.
- Walker2d-v4
  - REFLECT: Swap the observations dimensions of the left and right legs. The reward is unchanged.
- Ant-v4



(a) Ant-v4: ROTATE

(b) Ant-v4: REFLECT



(c) Humanoid-v4: ROTATE

Figure 15: Visualizations of some augmentations in MuJoCo environments.

  - REFLECT: Swap the observations dimensions of the front and back legs. *Unlike the other reflections, this augmentation affects the reward.* In particular, if the observed transition moves forward with velocity $\boldsymbol{v}$, the reflected transition will move backwards with velocity $-\boldsymbol{v}$. Thus, this reflection flips the sign of the “forward progress" reward term in the reward function.
  - ROTATE: Rotate the agent’s orientation by θ sampled uniformly at random from $[-\pi/6, \pi/6]$.
- Humanoid-v4
  - REFLECT: Swap the observations dimensions of the left and right arms/legs, and reflect torso joint angles, velocities, and orientation about the agent’s central axis. The reward is unchanged.
  - ROTATE: Rotate the agent’s orientation by θ sampled uniformly at random from $[-\pi/3, \pi/3]$.

F.3 AUGMENTED REPLAY RATIO

We repeat the same augmented replay ratio experiments detailed in Section 5.1.4 for MuJoCo tasks. We decrease the replay ratio β by generating more augmentations per observed transition while keeping the amount of augmented data used in policy/value function updates fixed, *i.e.*, we increase the augmentation ratio m while fixing the update ratio α. We consider all augmentations listed in Appendix F.2.

For the ROTATE augmentation, we fix the update ratio at $\alpha = 1$ and generate $m = 1, 2, 4$ augmented transitions per observed transition, corresponding to augmented replay ratios $\beta = 1, 0.5, 0.25$, respectively. Even though REFLECT can only generate a single augmented transition per observed transition, we can nevertheless study the augmented replay ratio as follows. Rather than generating an augmenting every observed transition, we instead augment transitions with some probability p such that the augmented replay ratio is $1/p$ in expectation. We can think of p as a fractional augmentation ratio. We fix the update ratio at $\alpha = 0.25$ and consider $p = 0.25, 0.5, 1$, again corresponding to augmented replay ratios $\beta = 1, 0.5, 0.25$, respectively.

Results are shown in Fig. **??**. A lower augmented replay ratio with REFLECT yields slight improvements in data efficiency for all environments considered. ROTATE improves data efficiency in Ant-v4. We observe much larger improvements with a lower augmented replay ratio in our core sparse reward tasks (Section 5.1.4).

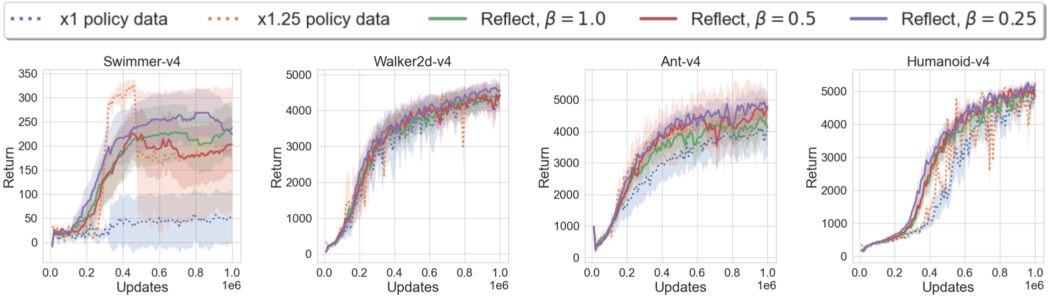


Figure 16: Lowering the augmented replay ratio for the REFLECT augmentation. Solid lines denote averages over 10 seeds, and shaded regions denote 95% confidence intervals.

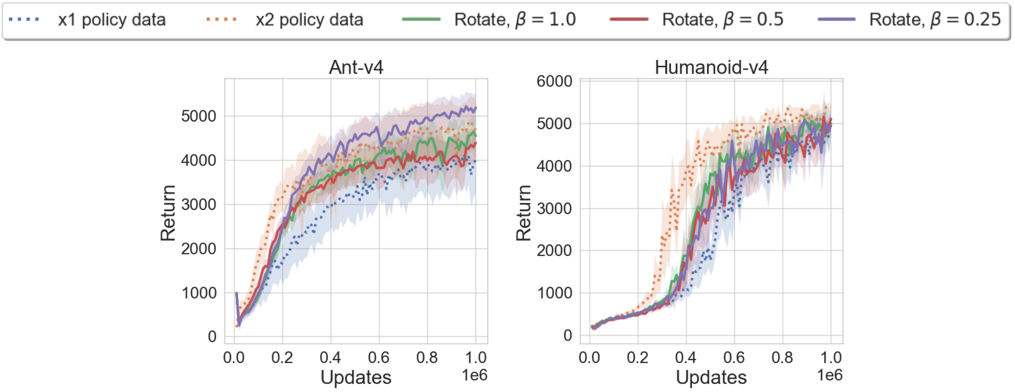


Figure 17: Lowering the augmented replay ratio for the TRANSLATE and ROTATE augmentations. Solid lines denote averages over 10 seeds, and shaded regions denote 95% confidence intervals.

G TRAINING DETAILS

We use the Stable Baselines3 (Raffin et al., 2021) implementation of DDPG (Lillicrap et al., 2015) and TD3 (Fujimoto et al., 2018) with modifications to incorporate augmentation into the RL training loop. In Panda tasks, we use DDPG since we found that it performs substantially better than TD3. This observation was also made by Gallouédec et al. (2021). All Panda experiments use the default hyperparameters presented in Table 1. These parameters are nearly identical to the those used by Plappert et al. (2018) and Gallouédec et al. (2021). The augmented replay ratio experiments in Fig. 6 and update ratio experiments in Fig. D.1 use different values for two hyperparameters specified below:

- Random action probability: 0
- Update frequency: Every timestep (observed replay ratio of 1)

We ran all experiments on a compute cluster using a mix of CPU-only and GPU jobs. This cluster contains a mix of Tesla P100-PCIE, GeForce RTX 2080 Ti, and A100-SXM4 GPUs. Due to limited GPU access, we only used GPUs for the augmented replay ratio experiments in Section 5.1.4, since these were our most computationally demanding experiments. We ran state-coverage, reward density, and generalization experiments on CPU only. CPU jobs took 12-36 hours each depending on the training budget, and GPU jobs took up to 16 hours each.