Extended Abstract

Motivation In real-world environments, agents often encounter scenarios that differ from their training distribution, which can lead to undesired behavior if the agent lacks mechanisms to recognize and respond to these unfamiliar states. Developing principled approaches to on-the-fly adaptation will help bridge the gap between simulation and deployment, ultimately making RL agents more reliable, generalizable, and applicable to complex real-world tasks.

Method To improve adaptation, we incorporate two novel mechanisms. First, we implement a discriminator-based recovery incentive: a neural network trained to distinguish familiar (in-distribution) states from novel ones. The agent is rewarded with $\lambda \cdot \log D(s)$, encouraging it to return to safer, previously seen regions. Second, we add an uncertainty penalty to the actor loss, discouraging high-variance actions by computing the variance across ensemble critics and subtracting $\lambda \cdot \text{Uncertainty}(s, a)$ from the Q-objective, with $\lambda = 0.01$.

Implementation We train all models with replay buffers that consist of 50% offline data and 50% online rollouts. Offline data is collected in the Pointmass environment by scripting motion so that the agent moves in the direction of the target state until success. Random perturbations are introduced in the start state for diversity of trajectories. We collect 1000 offline trajectories for use in conjunction with online data collected in the *Pointmass* environment with a 50 by 50 grid, start state of (7.5, 25.0), and target state of (42.5, 25.0). All evaluations occur in the *PointmassWind* environment, where a constant force in (0.0, 0.4) perturbs the agent in the positive y-direction at each step. For both UAR and DBR, we use $\lambda = 0.4$. We use an ensemble of 10 Q-networks, each of which is a two-layer MLP with 256 hidden units. We use the same architecture for our actor networks. All results are reported after 70,000 training iterations, which we found to be enough iterations for all training losses and accuracies to stabilize.

Results RLPD, lacking explicit recovery incentives, fails to reach the target state in the Pointmass-Wind environment, achieving -200 average reward (-1 reward for each of the maximum 200 steps in an episode that it isn't at the target). Uncertainty-Aware Recovery (UAR) and Discriminator-Based Recovery (DBR) both beat this baseline, achieving an average evaluation reward of -32.9 and -186.4 respectively. Qualitatively, the evaluation trajectories of the RLPD-trained agent attempt to resist the wind perturbation but ultimately still fail to reach the target. The DBR agent evaluation trajectories are similar to the RLPD agent trajectories, except the recovery incentive is strong enough and enables the DBR agent to occasionally achieve success. UAR is by the far most successful method, consistently reaching the target in a few steps. Qualitative observations of the evaluation trajectories indicate that the agent learns to dynamically resist the wind.

Discussion The superior performance of UAR over DBR can be attributed to its direct use of uncertainty estimates from the Q-function ensemble. This approach provides a more reliable recovery signal than DBR's distribution matching, which we found to be less effective due to the nature of our offline data generation. Our offline trajectories, generated with small random start state variations within a radius of 5 units, created a training distribution that made it difficult for the discriminator to provide strong recovery incentives for states far from the optimal trajectory. While our methods show promise in improving robustness to distributional shifts, several limitations should be noted. The current evaluation is limited to a simple 2D navigation task, and the effectiveness of these approaches in more complex environments remains to be tested. Additionally, our reliance on expert demonstrations may not be feasible in all real-world scenarios. Future work should explore the applications of these recovery mechanisms in higher-dimensional tasks and scenarios with suboptimal or limited demonstration data.

Conclusion In this work, we evaluate three offline-to-online reinforcement learning methods—RLPD, Discriminator-Based Recovery (DBR), and Uncertainty-Aware Recovery (UAR)—on a goal-reaching task in the Pointmass environment. We find that both DBR and UAR improve on the RLPD baseline in terms of robustness to an unexpected wind in the test-time environment, though UAR performs more consistently and with smoother trajectories. Further work is required to investigate transferability of these methods to other environments, the effect of random variation across training runs, and effectiveness when expert demonstrations are not available.

On-the-Fly Adaptation for Out-of-Distribution Robustness in Reinforcement Learning

Emily Liu
Department of Computer Science
Stanford University
emily712@stanford.edu

Flora Yuan
Department of Data Science
Stanford University
floray@stanford.edu

Nicole Tong
Department of Symbolic Systems
Stanford University
nwtong@stanford.edu

Abstract

In real-world environments, agents often encounter scenarios that differ from their training distribution, which can lead to undesired behavior if the agent lacks mechanisms to recognize and respond to these unfamiliar states. We aim to enable reinforcement learning agents to autonomously adapt to out-of-distribution (OOD) situations. We investigate two mechanisms for on-the-fly adaptation: uncertainty-aware recovery incentives and discriminator-based recovery incentives. Our uncertainty-aware approach penalizes actions in regions of high uncertainty, while our discriminator-based method provides recovery signals by distinguishing between offline and online state distributions. Evaluated on a Pointmass navigation task with unexpected wind perturbations, demonstrating that uncertainty-aware recovery incentives enable consistent goal-reaching behavior with smooth trajectories, achieving success in an average of 33.9 steps. Our discriminator-based approach shows promising but less consistent performance, while a baseline hybrid offline-online approach (RLPD) fails to adapt to unexpected conditions. These results highlight the value of explicit recovery mechanisms, particularly uncertainty-guided ones, for improve agent robustness under distributional shift. This project is significant because real-world deployment of reinforcement learning (RL) systems frequently involves unpredictable or partially observable environments where distributional assumptions made during training may not hold.

1 Introduction

The deployment of reinforcement learning (RL) agents in real-world environments presents a fundamental challenge: how can agents maintain robust performance when encountering scenarios that differ significantly from their training distribution? This distributional shift problem has emerged as one of the primary barriers preventing the widespread adoption of RL systems in safety-critical applications, where unexpected behaviors can have severe consequences. Traditional reinforcement learning approaches assume that the test environment closely matches the training distribution, an assumption that rarely holds in practice. Real-world environments are inherently dynamic, introducing novel obstacles, changing dynamics, sensor noise, and unforeseen perturbations that can cause well-trained agents to fail catastrophically. The problem becomes particularly acute in single-life scenarios, where agents must successfully complete tasks within a single trial without the opportunity for episodic resets or extensive online learning.

Stanford CS224R 2025 Final Report

Recent work has begun to address aspects of this challenge through various approaches. Single-Life Reinforcement Learning (SLRL) frameworks have proposed methods for leveraging offline prior data to improve single-trial performance, while systems like Vision-Language Model Predictive Control (VLM-PC) have demonstrated the potential of incorporating high-level reasoning for adaptive behavior selection. However, these approaches either rely heavily on pre-collected related data or require sophisticated language model infrastructure that may not be available across all RL domains. In this work, we present a comprehensive approach to on-the-fly adaptation for out-of-distribution robustness in reinforcement learning. In this work, we propose a lightweight, RL-native framework for on-the-fly adaptation to out-of-distribution (OOD) scenarios. We introduce and evaluate two novel recovery mechanisms that enable real-time correction during deployment: 1) Discriminator-Based Recovery (DBR), which rewards agents for staying close to offline expert trajectories using a learned classifier that distinguishes in-distribution from novel states, and 2) Uncertainty-Aware Recovery (UAR), which penalizes actions with high uncertainty based on ensemble Q-value variance.

2 Related Work

Prior work has also developed frameworks for adapting to unexpected scenarios. Single-Life Reinforcement Learning (SLRL) focuses on enabling agents to successfully complete a task within a single trial by leveraging offline prior data from related environments Chen et al. (2022). SLRL proposes distribution matching techniques, specifically, Q-weighted adversarial learning (QWALE)—to help agents make better use of prior experience. However, its approach primarily banks on the pre-collection of offline data from related environments and relies on aligning the distributions of these prior experiences with the test-time setting. Such prior data may, in reality, be incomplete, noisy, or unavailable for the exact test environment. Furthermore, QWALE and similar methods do not explicitly consider online, on-the-fly decision-making mechanisms during test time that allow an agent to dynamically select between strategies, prioritize exploration, or switch between multiple policies in response to uncertain events.

Another line of work relevant to our objectives is Vision-Language Model Predictive Control (VLM-PC) Chen et al. (2024), which leverages the reasoning capabilities of large vision-language models (VLMs) to enable legged robots to adaptively select behaviors in complex, real-world environments. VLM-PC addresses the challenge of navigating unpredictable and ambiguous scenarios—such as dead ends, debris, and cluttered spaces—by prompting VLMs to plan over sequences of high-level skills based on both the robot's visual observations and its history of prior interactions. This approach allows the robot to dynamically replan when progress toward a goal stalls, demonstrating effective on-the-fly adaptation without requiring environment-specific engineering or heavy human supervision.

While VLM-PC makes important progress toward robust adaptation in unfamiliar scenarios, its focus is primarily on leveraging pretrained VLMs for high-level reasoning and planning across skills, rather than developing mechanisms within the reinforcement learning framework itself. The system relies on explicit prompting and history conditioning to drive adaptation, rather than directly incentivizing the agent through mechanisms like uncertainty-aware value functions, policy selection strategies, or exploration incentives tied to distributional awareness. Our project aims to complement such high-level reasoning approaches by focusing on adaptive RL mechanisms at the policy or value-function level, allowing agents to autonomously adjust their behavior at test time even in the absence of language models or structured skill descriptions. This distinction positions our work toward broader applicability across RL domains where language supervision or structured skill libraries may not be available.

3 Methods

In order to improve robustness to out-of-distribution situations at test time, we build upon the RLPD (Reinforcement Learning with Prior Data) framework and investigate the effectiveness of (1) discriminator-based and (2) uncertainty-aware recovery incentives.

3.1 Problem Setup

In order to train and evaluate agents for out-of-distribution robustness, we utilize two environments: **Pointmass** and **PointmassWind**. The Pointmass environment consists of a 50 by 50 empty grid,

wherein the agent must navigate from a given start state, (7.5, 25.0) to a target state (42.5, 25.0). This environment is used for offline data collection and online training for all methods. The PointmassWind Environment is identical to the Pointmass Environment except that at each environment step, a random "wind" is added to the y axis of the new agent state, uniformly sampled from the interval [0, wiggle_weight], where wiggle_weight is a tunable hyperparameter. In our experiments, we choose wiggle_weight=0.4, for continuity with prior works Chen et al. (2022) and because empirically we found it to be an appropriate level of difficulty for the complexity of our networks. The action spaces for both environment is a continuous vector ranging from [-2, 2] in each axis. The PointmassWind environment is never encountered during training for any of the methods, and is used for solely during evaluation to simulate an "out-of-distribution" scenario. We tested several different grid sizes and found the 50 by 50 grid task to be an appropriate level of difficulty for the scale of our networks and compute.

3.2 Baseline Algorithm: RLPD

As a baseline, we use a standard RL method called Reinforcement Learning with Prior Data (RLPD), which combines offline data with standard off-policy online learning methods. RLPD uses Soft Actor-Critic (SAC) with an ensemble of 10 Q-networks, trained on a replay buffer consisting of 50% offline data and 50% online rollouts. Gradient propagation occurs only after 5000 iterations, ensuring that updates begin only after sufficient online data is collected. The original RLPD method includes a pretraining step, but we omit this due to the lack of a large amount of offline data, and choose to incorporate our small offline dataset solely through the joint offline-online training. After T_{start} steps, the agent begins updating from a combined buffer $\mathcal{B} = \mathcal{D}_{\text{demo}} \cup \mathcal{D}_{\text{online}}$. Additional differences between RLPD and standard SAC include the incorporation of layer-norm in the critic networks to avoid catastrophic overestimation.

3.3 Discriminator-Based Recovery Incentives (DBR)

The first method that we propose is a Discriminator-Based Recovery incentive. Discriminator-Based Recovery Incentives (DBR) is a way to address the challenge of policy recovery from out-of-distribution states in offline-to-online reinforcement learning. The core idea is to learn a discriminator D_ϕ that distinguishes between states from the offline dataset (positive samples) and states encountered during online interaction (negative samples). The discriminator provides a recovery signal that guides the policy back to the offline distribution when it encounters unfamiliar states.

The recovery reward is computed as: $r(s) = log(D_\phi(s))$

This reward formulation provides a strong gradient for the policy to move towards offline distribution, as states that are more similar to the offline data receive higher rewards. For the advanced variant, we employ a GAIL-style reward: $r(s) = -log(1 - D_\phi(s))$

We implemented two variants of the discriminator:

1. **Simple Discriminator** The basic implementation consists of a neural network D_ϕ with the following architecture:
    - Input Layer: State observation space obs_dim
    - Hidden Layer: 128 units with ReLU activation
    - Output Layer: Single unit with sigmoid activation
2. **Advanced Discriminator** Since we observed very little change from our baseline RLPD, we implemented an enhanced version that introduced several key improvements:
    - **State-Action Discriminator**: Instead of just a State-based Discriminator, we implemented a state-action discriminator $D_\phi(s, a)$ that considers both the state and the action taken. This model accepts an input of dimension of obs_dim and action_dim, formed by concatenating the state and action vectors. By including action information, this architecture is more powerful in capturing policy-specific behaviors and can better distinguish between different strategies that might lead to the same state.
    - **Mixup Regularization**: To improve the generalization ability of the discriminator, we implement mixup regularization by generating interpolated samples between offline and online states. Specifically, we sample a mixing coefficient λ from a Beta(α, α)

distribution and use it to create mixed inputs of the form $x_{\text{mixed}} = \lambda x_1 + (1-\lambda)x_2$ and corresponding mixed labels $y_{\text{mixed}} = \lambda y_1 + (1-\lambda)y_2$. This technique encourages the discriminator to learn smoother decision boundaries, reduces overconfidence in predictions, and improves robustness to distributional shifts and adversarial examples.

- **Q-weighted Loss**: To prioritize learning from more experiences, we implement a Q-weighted loss function that incorporates value estimates into the discriminator's training. Specifically, the binary cross-entropy loss is weighted by the Q-value of each sample, resulting in $\mathbb{E}[Q(s) \cdot \text{BCE}(D_\phi(s), y)]$. This encourages the discriminator to focus on high-value states, aligning better with the policy's objectives and enabling more efficient use of training data.

The advanced discriminator is trained using binary cross-entropy loss: $\mathcal{L}(\phi) = \mathbb{E}_{s\sim p_{\text{offline}}}[\log D_\phi(s)] + \mathbb{E}_{s\sim p_{\text{online}}}[\log(1 - D_\phi(s))]$

This approach aims to provides imitation-driven reward shaping, which is especially beneficial in sparse reward environments where external feedback is limited or delayed. By encouraging the agent to return to regions of the state space that resemble offline expert demonstrations, the discriminator acts as a surrogate reward function that reinforces safe and familiar behaviors. This helps mitigate the risk of divergence when the agent encounters unfamiliar or low-reward regions during online fine-tuning.

3.4 Uncertainty-Aware Recovery Incentives (UAR)

We also propose a second method, called Uncertainty-Aware Recovery Incentives (UAR), which involves introducing an **uncertainty-aware recovery penalty** directly into the actor loss function.

We estimate uncertainty as the variance of the predicted Q-values across an ensemble of N critics:

$$u(s,a) = \text{Var}(Q_1(s,a), Q_2(s,a), \ldots, Q_N(s,a))$$

In our experiments, we use $N = 10$. The actor loss is then augmented with an uncertainty penalty term:

$$\mathcal{L}_{\text{actor}}^{\text{UAR}} = \mathcal{L}_{\text{actor}} + \lambda_{\text{uar}} \cdot u(s,a)$$

where λ_{uar} is a hyperparameter controlling the strength of the penalty. In our experiments, we find $\lambda_{\text{uar}} = 0.01$ to be the most effective value, so all UAR results are reported with $\lambda_{\text{uar}} = 0.01$.

Through this uncertainty penalty in the actor loss, we aim to discourage the agent from selecting actions in regions of the state-action space where the value function exhibits high epistemic uncertainty. The agent is incentivized to return to states that it is more familiar with, or the states closer to the expected optimal trajectories that it saw under the training environment without unexpected wind.

3.5 Offline Data Generation

We generate expert offline trajectories data using scripted navigation: given a start s_0 and goal g, we move directly toward the goal with fixed magnitude η:

$$a_t = \eta \cdot \frac{g - s_t}{\|g - s_t\|}$$

with $\eta = 2$. In order to further encourage robustness to states outside of the optimal direct horizontal trajectory, we generate varying start and end locations by uniformly sampling a random offset between -5 and 5 to add to our default start point on the grid. These trajectories are stored for use as offline data in each of the three methods: RLPD, Discriminator-Based Recovery Incentives, and Uncertainty-Aware Recovery Incentives, as they all use a hybrid offline-online approach.

4 Experimental Setup

4.1 Training

We train all models in the Pointmass environment with a 50 by 50 grid. We limit each episode (in both training and evaluation) to a maximum of 200 steps. See section 3.1 for a more detailed description of the Pointmass environment.

We generate 1000 offline trajectories according to the procedure described in section 3.5. Each trajectory includes observations, actions, rewards, and next observations. For our SAC implementation, we use the following network architecture: The actor network consists of a two-layer MLP with 256 hidden units per layer and ReLU activation functions. The output layer uses a TanhNormal distribution to generate continuous actions. The critic network is implemented as an ensemble of ten Q-networks, each with the same architecture as the actor (256 hidden units per layer). We use Adam optimizer with a learning rate of 3e-4 for both actor and critic networks. We use a discount factor of 0.99 and a soft update coefficient of 0.005 for the target networks. We have two versions of the discriminant (Simple and Advanced); the architecture of both versions of the discriminant is described in section 3.3. All results are reported after 70,000 training iterations, which we found to be enough iterations for all training losses and accuracies to stabilize.

4.2 Evaluation

We evaluate all models in the PointmassWind environment with a wiggle weight of 0.4, which simulates an unexpected "out-of-distribution" scenario at test time that was never experienced during training. See section 3.1 for a more detailed description of the PointmassWind environment.

We quantitatively evaluate our methods using the following metrics:

- **Average Episode Return:** Average reward per episode over 10 episodes.
- **Trajectory Efficiency:** Average number of steps required to reach the goal, over 10 episodes.
- **Training Efficiency:** Number of training steps required before a successful evaluation trajectory.

Additionally, we visualize the training and evaluation trajectories for qualitative analysis.

5 Results

5.1 Quantitative Evaluation

We quantitavely evaluate each of our methods with the metrics describes in Section 4.2. Results are reported in Table 1. We observe that Uncertainty-Aware Recovery Incentives (UAR) is the first method to successfully complete the evaluation task: reaching the target state even when subjected to unexpected wind. UAR reaches the target state in an average of 33.9 steps, and first achieves a success in the PointmassWind evaluation environment after 25,000 training steps. Discriminator-based recovery incentives (DBR) is the second method to achieve success in the evaluation environment, completing the evaluation task successfully at least once after 24,000 training steps. After 70,000 training steps, it exhibits an average evaluation reward of -186.4, indicating that it successfully reaches the target state on a few of the 10 evaluation episodes, but failed on other episodes. RLPD fails to successfully reach the target state when subject to unexpected wind at all, resulting in an average reward of -200, as it incurs a negative reward of -1 for each of the maximum 200 steps per episode that it is not at the target state.

5.2 Qualitative Analysis

By visualizing rollouts during both training and evaluation, we also qualitatively compare the performance of RLPD, UAR, and DBR.

Training Trajectories In Figure 1, we can see that all three methods successfully reach the goal state in the Pointmass environment with pretty similar trajectories. The overall trajectory shapes are similar, suggesting that each agent is able to learn the nominal behavior required for the task in a

Table 1: Quantitative metrics for RLPD, UAR, and DBR

Method	Avg. Reward	Avg. Eval Steps	Train Steps until Success
RLPD	-200	200	-
Uncertainty-Aware	-32.9	33.9	25K
Discriminator-Based	-186.39	186.5	24K

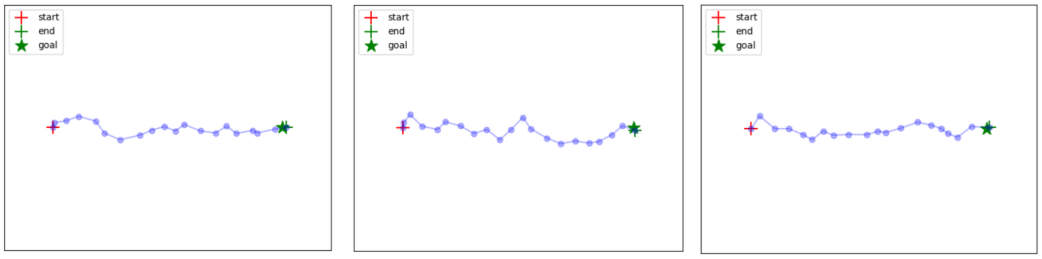

Figure 1: Visualization of online training rollouts in the **Pointmass** environment for each of our three methods. Left: RLPD, Middle: Uncertainty Aware Recovery Incentives, Right: Discriminator based recovery incentives.

low-noise, in-distribution setting. All three methods take a relatively direct and straight path from the start state to the goal state, though UAR (middle panel) seems to exhibit slightly more deviation from the optimal horizontal path than the others, occasionally taking actions with larger components in the y-direction. It is unclear why UAR exhibits this instability in training trajectory while the RLPD and DBR agents show slightly more deterministic and flattened paths. Further experimentation is required to determine the root cause of this variation.

Evaluation Trajectories In Figure 2, we see the behavior of RLPD, UAR, and DBR diverge. The RLPD-trained agent (left panel) is blown steadily upward by the wind in a linear fashion until overshooting the target in the x-direction. The agent then attempts to correct by suddenly moving downward, but appear to lack a strong enough recovery incentive to make it all the way to the target state. The DBR-trained agent (right panel) is also blown steadily upward by the wind, but appears to resist the upward wind slightly more than the RLPD agent, as it exhibits a slightly less than linear trajectory. However, it also exhibits similar behavior to RLPD in that the agent blows steadily upward until reaching the target x-coordinate, at which points it consistently tries to move directly vertically downwards until it eventually reaches the target. In contrast to both of these methods, UAR exhibits a much smoother recovery trajectory. It exhibits a relatively horizontal or gently downward sloping trajectory after the first 5 steps, likely because it takes actions to counteract the unexpected wind earlier in the trajectory. Overall, UAR seems to exhibit the most smooth and optimal trajectory under the unexpected wind. DBR exhibits less smooth behavior but ultimately strong enough recovery incentives to reach the target state. RLPD demonstrates impressive learned behavior to move toward target given that it had no explicit recovery incentives, but ultimately fails to reach the target state.

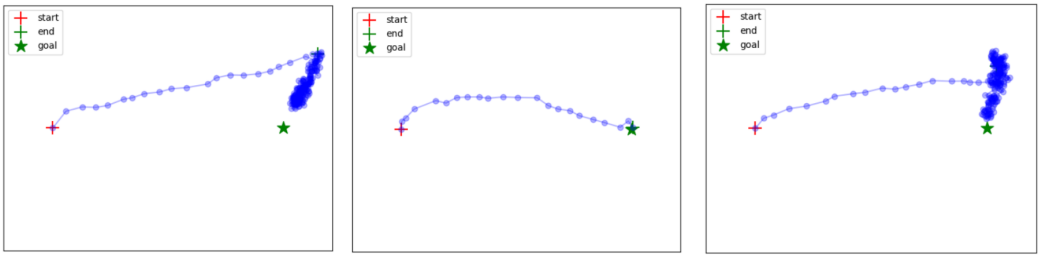

Figure 2: Visualization of evaluation rollouts in the **PointmassWind** environment for each of our three methods. Left: RLPD, Middle: Uncertainty Aware Recovery Incentives, Right: Discriminator based recovery incentives.

6 Discussion

We demonstrate that our two proposed methods both improve upon the RLPD baseline in terms of robustness to unexpected scenarios at test-time like a wind perturbance. Discriminator-based recovery (DBR) manages to achieves a few successful episodes per 10 episodes, with an average evaluation reward of -186.4. Uncertainty-aware recovery (UAR) consistently reaches the target state even under an unexpected wind, reaching the target in 34 steps on average.

It is expected that RLPD completely fails the PointmassWind task, as it has no explicit recovery incentives or training signal to teach it what to do under this unexpected circumstance. The fact that it tries to move downward toward the correct target at all is surprising, and indicates that the offline expert demonstrations likely encoded a very strong understanding of the exact target coordinates in our actor network. While DBR performed better than RLPD, occasionally reaching the target state in the evaluation environment, it is surprising that DBR didn't exhibit larger gains over the RLPD baseline, while UAR drastically improved upon both DBR and UAR. We hypothesize that this is because our trained discriminator will be more certain in its classification of states slightly offset from the optimal flat horizontal trajectory as offline states, rather than states exactly on the optimal horizontal trajectory. This is because of our random variation of the start state within a circle of radius 5 centered at the original start state for the offline data, so states slightly offset from horizontal will be the states most commonly seen in the offline trajectories. This could explain why DBR exhibits a relatively weak recovery incentive in the first half of each trajectory: these states have not yet deviated too far from states seen in our given offline trajectories.

While these results are very promising and indicate potential of our methods to improve robustness of agents to unexpected scenarios at test time, there are several limitations to our work. Firstly, during experimentation we observed slight variations in success between training runs. Future work should consider setting random seeds and averaging results across seeds to decrease the variance of results. Additionally, we evaluated on only a single task: a simple Pointmass environment that requires moving from start to target across an empty board. Further work is required to evaluate how our methods perform on non-2D Pointmass navigation environments, such as robot arm movement, humanoid movement, or autonomous driving. It is also important to note that our offline data consists of expert data, wherein each trajectory always successfully reaches the target state. This is possible in our simple setup where both the target state and the optimal trajectory to reach it are known, but this type of expert data may not be available in all cases. It would be beneficial to evaluate the effectiveness of our proposed methods when expert demonstrations are not available. Further, it is possible that more comprehensive tuning of hyperparameters (maximum episode steps, offline to online data ratio, lambdas, etc.) could yield better results for our methods.

7 Conclusion

In this work, we find that penalizing variation in predicted Q-value among critics (Uncertainty-Aware Recovery) and augmenting rewards at each state with predicted likelihood of aligning with offline states (Discriminator-Based Reward), helps improve robustness to out-of-distribution situations at test time compared to baseline soft-actor-critic methods. Uncertainty-aware recovery (UAR) in particular is very effective, enabling the agent to reach the target state in 34 steps on average. Qualitatively, UAR also exhibits smooth stable trajectories. However, further work is required to investigate transferability of these methods to other environments, the effect of random variation across training runs, and generalization to situations where expert demonstrations are not available. However, overall, our proposed methods show promising potential in improving robustness of agents to out-of-distribution states at test time.

8 Team Contributions

- **Group Member 1: Emily Liu** Implemented pointmass and pointmasswind environments, uncertainty-aware recovery incentives, and offline trajectory generation. Set up RLPD, ran experiments. Contributed to milestone and writing abstract, methods, results, discussion, conclusion of final report.

- **Group Member 2: Flora Yuan** Implemented discriminator-based recovery incentives and ran experiments. Set up RLPD. Contributed to proposal, milestone and methods, results, abstract, introduction, and related work of final report.
- **Group Member 3: Nicole Tong** Implemented offline data generation. Contributed to proposal and writing abstract, introduction, related work, methods of final report.

Changes from Proposal Our original proposal stated that it would explore three essential mechanisms: uncertainty-aware value guidance, contextual policy selection, and in-distribution recovery incentives. We implemented the first and third of these, but ended up not exploring the second method.

References

Annie S. Chen, Alec M. Lessing, Andy Tang, Govind Chada, Laura Smith, Sergey Levine, and Chelsea Finn. 2024. Commonsense Reasoning for Legged Robot Adaptation with Vision-Language Models. arXiv:2407.02666 [cs.RO] https://arxiv.org/abs/2407.02666

Annie S. Chen, Archit Sharma, Sergey Levine, and Chelsea Finn. 2022. You Only Live Once: Single-Life Reinforcement Learning. arXiv:2210.08863 [cs.LG] https://arxiv.org/abs/2210.08863

A Additional Experiments

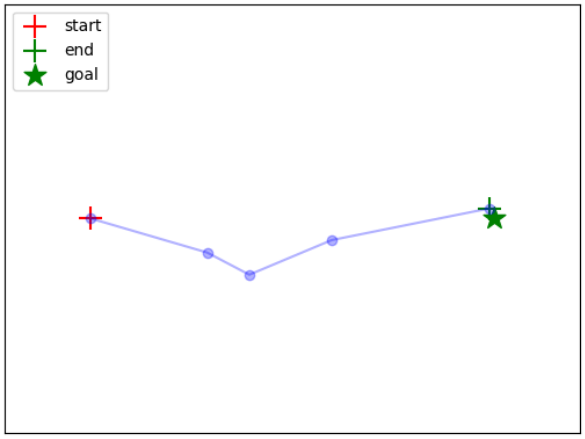

Figure 3: Visualization of online training rollouts in the **Pointmass** for RLPD on a 10x10 Environment at 40000 steps. We found this environment to be too simple.

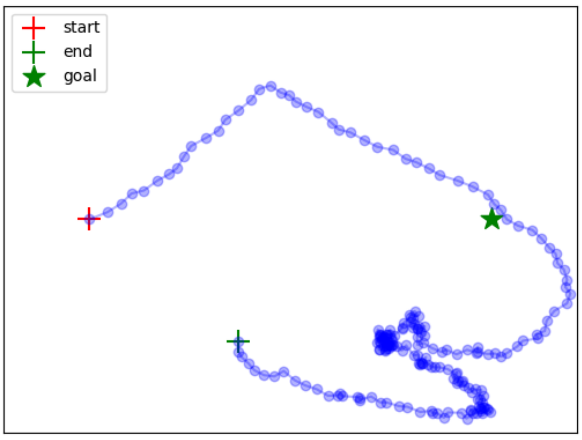

Figure 4: Visualization of online train rollouts in the **Pointmass** for RLPD on a 100x100 Environment at 130000 steps. We found this environment to be too difficult.