Cold Diffusion on the Replay Buffer: Learning to Plan from Known Good States

Anonymous Author(s)
Affiliation
Address
email

Abstract: Learning from demonstrations (LfD) has successfully trained robots with impressive generalization capabilities. However, many of the most powerful techniques for imitation do not ensure that generated robot behaviors are actually feasible. In this work, we explore the problem of ensuring the feasibility of plans generated by LfD. Like prior work, we use a temporal diffusion model with pinned start and goal states to facilitate imitation via in-painting. Unlike prior work, we use cold diffusion to force the entire optimization process to be routed through the agent's replay buffer of previously visited states. This routing procedure ensures that the final generated trajectories lie entirely in the feasible region of the robot's state space. We test this procedure on simulated robotic environments where obstacles are present, and see it greatly improves the agent's ability to avoid these obstacles during planning. Happily, we find that this procedure improves the final performance of LfD even when the agent encounters no obstacles limiting feasibility. We hypothesize that this is due to the decreased transport cost through the optimization space when learning over the buffer, and carry out an analysis to better understand this phenomenon.

1 Introduction

Consider a robot in a large enclosed garden. At the center of this garden, there is a rock. To initialize the robot's behaviors, humans collect some demonstration data of the robot doing various tasks around the garden e.g. pulling weeds, watering plants, pruning hedges. During these demonstrations, humans naturally avoid the large rock at the garden's center, preferring to walk around it. At deployment time, the robot is able to accomplish the demonstrated tasks successfully. However, in one instance, the robot's start location changes slightly. Trying to generalize to a new situation, the robot draws up a new plan that unfortunately sends it barrelling straight into the rock. The robot has fallen victim to faulty generalization. How should an agent know what paths are okay for generalization, and which are problematic?

Recent techniques for LfD have shown great promise in leveraging diffusion models to facilitate planning. In Diffuser [1], a robot's start position and goal position are fixed, and the rest of the trajectory is filled in with a temporal diffusion model. This procedure is sometimes called in-painting. A major advantage of in-painting with diffusion is that it allows for planning over an entire trajectory. Rather than auto-repressively considering one planned state at a time, in-painting allows agents to generate plans with temporal coherence across all planned states.

One drawback of diffusion planners is that the algorithm's prior is a d-dimensional Gaussian space. In the context of imitation, this means the algorithm starts with an assumption that all intermediate states between the start and the goal can occur anywhere on a d-dimensional grid. This can be a problematic assumption if the robot's true state space has holes or infeasible regions. While the diffusion process should in theory pass through these regions and land on a feasible trajectory that follows the data distribution, we find in practice that this is often not the case. Small changes in

Submitted to the 7th Conference on Robot Learning (CoRL 2023). Do not distribute.

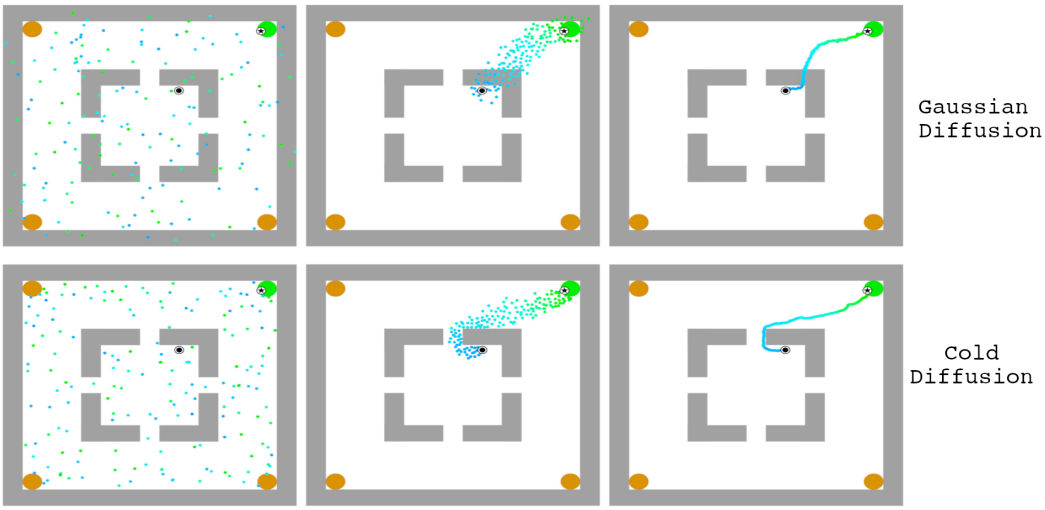


Figure 1: Cold Diffusion on the Replay Buffer: In contrast to Gaussian diffusion, we define a desegregation operator that iteratively maps a trajectory's points to randomly sampled points on the agent's replay buffer. We then learn a restoration operator $\mathcal{R}$ that reverses this noising process. At test time, the agent randomly samples points on the replay buffer and then runs the diffusion process backwards to obtain a trajectory that completes the desired task. Because this entire diffusion process happens on the replay buffer, the resulting trajectory is guaranteed to be feasible.

the robot's start or goal position will often generate trajectories that attempt to pass through these infeasible regions, in much the same way a garden robot might accidentally crash into a rock.

In this paper, we investigate the problem of using an agent's past experience to help guide a diffusion planner's optimization. Our proposed algorithm, Cold Diffusion on the Replay Buffer (CDRB), begins with a replay buffer of past trajectories. Iteratively, we apply a noising process that maps states along the trajectory to nearby points in the replay buffer. As this process is continually applied, the original trajectory degrades, transforming onto a set of normally distributed nodes across the replay buffer. Using cold diffusion, we train a planner to reverse this noising process, iteratively mapping these noisy nodes back onto the original trajectory. At test time, we can accomplish new goals by pinning a start and an end position and applying the cold diffusion process to generate a feasible path through the replay buffer.

The rest of this paper will be presented as follows.

- We will cover the necessary background on diffusion, cold diffusion, and diffuser.
- We adapt the diffusion process to be routed entirely though the replay buffer, ensuring that final learned plans lie entirely in the robot's feasible space. A concrete algorithm, Cold Diffusion on the Replay Buffer (CDRB), is presented.
- Experimental results are presented on a variety of simulated robotic domains, including settings where the robot faces significant obstacles blocking its path. In these settings, we see that CDRB plans more efficient paths. Somewhat surprisingly, we also see that CDRB is more performant than Gaussian diffusion even when no obstacles are present.
- An analysis is carried out to shed light on the average length of trajectories generated under Gaussian diffusion and CDRB. This analysis suggests that, in general, CDRB converges to smoother paths, which helps it achieve higher success rates in a variety of environments.

2 Related Work

Learning from Demonstrations (LfD) is a cornerstone of robotics [2, 3]. The density matching approach considered in this paper was pioneered by maximum entropy formulations of apprenticeship learning [4, 5, 6, 7]. GAIL [8] accomplishes density matching by using a GAN-like discriminator [9]

as a reward function. This formulation was later extended to general f-divergence metrics [10, 11]. Constraining this f-divergence optimization process was considered in Durugkar et al. [12], Ma et al. [13]. Meanwhile, [14, 15] showed how to use hindsight relabeling and Q-learning in the context of imitation learning. Safe imitation learning has previously been considered in [16], which uses ensembles to correct learned policies mid-rollout, as in Dagger. See also Yin et al. [17] for a general overview, and Chen et al. [18] for thoughts on safe imitation in the context of self-driving.

Diffusion Models have enjoyed recent success in domains such as image generation, video generation, text-conditional image generation, and image editing [19]. In robotics specifically, Janner et al. [1] learns a diffusion process to generate entire trajectories conditioned on start and end state information. Later, Pearce et al. [20] provided several techniques for improving the training process, including the use of transformers for learning the noise schedule. Finally, Wang et al. [21] considers the problem of learning multi-modal action distributions with diffusion models. We use Janner et al. [1], Pearce et al. [20], Wang et al. [21] as baselines in Section 5.

Safe Planning and Exploration is a perennial problem in the RL and control literature, tracing its modern roots back to constrained MDPs [22]. The notion of learning that obeys feasible state constraints was later crystallized in [23], though this approach requires human labels. Classic trajectory optimization literature at times maintains feasibility by recasting the problem as sequential convex optimization with collision checking [24]. The planning literature often considers feasibility-constrained optimization [25]. Work on the Bug Trap domain, where the agent has to escape a narrow corridor, was particularly inspiring for our current work [26]. Modern work has sought to bridge modern RL and safe planning, often by optimizing stability [27, 28], reachability [29], or reversibility [30] to encourage safe behaviors. On a final note, a large number of works try to infer safety directly from human preferences [31, 32, 33, 34]. See Pecka and Svoboda [35], Garcıa and Fernández [36] for comprehensive literature reviews of safety in RL.

3 Background

Learning from Demonstrations (LfD) is a general problem setting where a robotic system learns to copy a demonstrated behavior. Suppose we start with some demonstration trajectory $\tau = [s_0, a_0, s_1, a_1, \ldots, s_T, a_T]$. In the action-density matching variant of LfD, we assume there exists an underlying conditional action density $\mathcal{D}_\tau(a_i|s_i, s_T)$. A policy π is then trained to recover this distribution. In other words, we train π to recover τ's conditional state-action distribution. Meanwhile, in the state-density matching variant of LfD, we train π such that $D_{KL}(\rho_\pi(s|s_i, s_T)||\rho_\tau(s|s_i, s_T))$ is minimized. In this variant, we only care about state-density matching, and action choice is optimized only to facilitate this matching.

The recently proposed Diffuser algorithm is an interesting case, because it considers the problem of density matching over an entire trajectory's state-action pairs. Given s_0, a_0, s_T, a_T, Diffuser attempts to recover all intermediate state-action pairs simultaneously. Let $\tilde{\tau} = [s_0, a_0, z_{1,s}, z_{1,a}, \ldots, s_T, a_T]$, a trajectory with a desired start and end point but randomly sampled Gaussian noise in the middle. The Diffuser objective is to learn a map $R : \tilde{\tau} \to \tau$, minimizing the loss function $D_{KL}(\rho(R(\tilde{\tau}))||\rho(\tau))$.

Given its name, it should come as no surprise that Diffuser accomplishes this density matching by leveraging diffusion models. Recently popularized, diffusion models can be thought of as a generative model that learns to iteratively map some noisy input onto a target distribution. In the context of LfD, we are going to take some noisy intermediate states $z_{1,s}, z_{1,a}, \ldots z_{T-1,s}, z_{T-1,a}$ and iteratively denoise them to recover $s_1, a_1, \ldots, s_{T-1}, a_{T-1}$. Assume we start with a noisy trajectory $\tau(t)$, where t indexes the diffusion process timestep, not the trajectory time. The trajectory can be expressed as

$$\tau(t) = \alpha(t)\tau(0) + \sigma(t) \cdot z, z \sim \mathcal{N}(0, I),$$

where p_t is the pdf of $\tau(t)$, $\sigma(t)$ scales the noise magnitude at timestep t, and $\alpha(t)$ scales based on the data magnitude.

The LfD problem is then to gradually remove the noise from $\tau(t)$ and recover $\tau(0)$. This can be done with using a reverse SDE that flows backwards from $t = 1$ to $t = 0$, at each step making use of a

noise-perturbed score function $\nabla_\tau p_t(\tau)$. For example, if we assume Gaussian noise is added to the trajectory at each timestep, then the sampling procedure can be defined as

$$d\tau(t) = \left[-\frac{d[\sigma^2(t)]}{dt}\nabla_x \log p_t(x)\right] dt + \sqrt{\frac{d[\sigma^2(t)]}{dt}} dw$$

with dw a Wiener process. Our goal is to learn the noise-perturbed score function $\nabla_\tau p_t(\tau)$. This can be done with denoising score matching. In particular, let $s_\theta(\tau(t), t)$ be the parameterized score model. The goal is to then learn the noise scale at time t

$$L_t = E_{\tau(0)\sim\rho(\tau), z\sim N(0,I)}\left[\|\sigma_t s_\theta(\tau(t), t) - z\|_2^2\right]$$

Where $\rho(\tau) = p_{data}$ is the demonstration density. During training, we consider a weighted sum over L_t, trying to learn the appropriate noise level across various points in time. See [37] and [38] for more discussion on possible weighting strategies. At test time, new trajectories can be generated by fixing s_0, s_T, sampling $\tau(0) \sim \mathcal{N}(0, \sigma^2)$, and iteratively applying the Euler-Maruyama method with the learned $s_\theta(\tau(t), t)$ to recover $\tau(0)$

$$\tau(t) = \tau(t + \Delta t) + (\sigma^2(t) - \sigma^2(t + \Delta t))s_\theta(\tau(t), t) + \sqrt{\sigma^2(t) - \sigma^2(t + \Delta t)}z$$

Note that s_0, s_T are held fixed during this process. This is referred to as *pinning*.

In contrast to diffusion, the recently proposed cold diffusion does not require a random degradation process. Instead, a score function can be learned with respect to any degradation process $D(\tau(0), t)$ that transforms the data from $\rho(\tau(0))$ to $\rho(\tau(t))$. We write the cold diffusion procedure iteratively as

$$\tau(t-1) = \tau(t) - D(\tau(0), t) + D(\tau(0), t-1)$$

This mapping is learned by learning a deterministic noise schedule and applying DDIM [39]. See Bansal et al. [40, Appendix A.6] for more details.

4 Methodology

We start by considering the Gaussian diffusion process in a simple grid environment, pictured in Figure 2. In this environment, the agent starts at some random position s_0 in the grid and must learn to navigate to one of the 4 target goal positions s_T while avoiding the walls. The diffusion process samples noisy intermediate states from a Gaussian $\tau(t) = z_1, z_2, \ldots, z_{T-1} \sim p(t)$. We then reverse the noising process to obtain a planned trajectory $\tau(0) = s_1, s_2, s_3, \ldots, s_{T-1}$ connecting s_0 and s_T.

If the entire region connecting s_0 and s_T is feasible, then the above diffusion process presents no issues. However, what happens when there is an infeasible region between the two points, for example a wall? Since the Gaussian score function at time t is completely divorced from the realities of the environment, it is likely to sample intermediate points z_i from the infeasible region, e.g. it will sample a point inside the wall. We hope that reversing the diffusion process will fix this issue and generate a feasible trajectory. However, in practice we often find that the agent tries to generalize incorrectly and draw straight line paths across an infeasible region, especially when s_0 and s_T were not present in the training data.

Perhaps the simplest fix to this ineffability issue is by projecting the fully denoised trajectory $\tau(0)$ back onto the agent's feasible region. This can be accomplished with minimal overhead by keeping a replay buffer $\mathcal{B} = (\mathbf{s}, \mathbf{a})$ that contains all state-action pairs present in the expert demonstrations τ. We then project each trajectory point $s_i(0), a_i(0) \in \tau(0)$ onto the closest replay buffer point

$$\underset{s^*, a^* \in \mathcal{B}}{\arg\min} \|s^* - s_i(0)\| + \|a^* - a_i(0)\| \tag{1}$$

Such a feasibility projection is intuitively satisfying. However, if we think about it, we will soon realize that such a projection won't work in our simple grid environment. The issue is that the entire planned trajectory is attempting to go through the wall. Projecting this trajectory back onto the replay buffer will simply cluster all the infeasible points on either side of the wall. If the agent tries to follow

this plan, it will naively go towards the wall and get stuck on one side, rather than correctly planning to avoid the wall entirely.

Nevertheless, we like the idea of this feasibility projection. Thus, we seek to incorporate it into the diffusion process itself, ensuring the entire optimization procedure is routed through the replay buffer. In particular, we can define a degradation process that iteratively maps points $s(0), a(0)$ onto points selected from the replay buffer $s(t), a(t) \sim \mathcal{B}$. As t increases, the maximal distance from $s(0), a(0)$ to $s(t), a(t)$ also increases. Cold diffusion can then be used to reverse this degradation process by starting from a random sample of points on the replay buffer $\tau(t) \sim \mathcal{B}$, and learning to iteratively map them back onto a trajectory $\tau(0)$ that starts at s_0 and reaches s_T. This procedure is described in detail below.

Forward Process: Degradation

We want to define a $D(\tau(0), t) = \tau(t)$ that takes points in a trajectory and iteratively spreads them out across the replay buffer. We want this degradation to vary continuously in t, with larger values of t leading to sampling points that are more spread out across the buffer. We can consider this process pointwise for each $s \in \tau$. Intuitively, at time k, we want to sample some point from the replay buffer that is within some small epsilon ball about s

$$p_k \sim B_{\epsilon_k}(s) \cap \mathcal{B} \tag{2}$$

We then replace point s from the trajectory τ with the point p_k from the replay buffer, $D(s_0, k) \to p_k$. As k increases, ϵ_k will also increase, which will lead to more degradation as the sampled points stray further from the original source s. We consider several schedules for increasing the size of ϵ_k in the experimental section. For example, in a linear distance schedule, some max ball size is fixed based on the radius of the environment, $d_{\max}$. Then ϵ_k varies linearly from 0 to $d_{\max}$

$$\epsilon_k = \frac{k}{t} \cdot d_{\max}$$

If we always applied this operator starting from s, there would only be a weak sense of continuity among degradation steps, since the process at step $k+1$ would have no memory of the previously sampled $p_k \sim B_{\epsilon_k}(s)$. In practice, we achieved better performance by iteratively applying the operator to the previously sampled points.

$$\begin{aligned} p_0 &\sim B_{\epsilon_0}(s) \cap \mathcal{B} \\ p_1 &\sim B_{\epsilon_1}(p_0) \cap \mathcal{B} \\ p_k &\sim B_{\epsilon_k}(p_{k-1}) \cap \mathcal{B} \end{aligned}$$

Backward Process: Restoration

The reverse process involves the restoration operator $\mathcal{R}$, which (approximately) inverts the degradation operator D. This operator has the property that

$$\mathcal{R}\left(\tau(t), t\right) \approx \tau(0)$$

In practice, this operator is implemented via a neural network parameterized by θ. The restoration network is trained via the minimization problem

$$\min_{\theta} \mathbb{E}_{\tau \sim \rho(\tau)} \left\| \mathcal{R}_\theta(D(\tau, t), t) - \tau \right\|$$

where τ denotes a random trajectory sampled from distribution $\rho(\tau)$ and $|\cdot|$ denotes a norm. This restoration process aims to recover the original trajectory from the degraded one. In particular, $D(\tau, t)$ degrades the trajectory to $\tau(t)$, and then $\mathcal{R}_\theta(\tau(t), t)$ attempts to reverse the degradation and recover $\tau(0) \approx \tau$. At test time, the learned restoration operator is used for generating feasible trajectories with entirely in-distribution states. During restoration, the start state s_0 and goal state s_T are pinned. Full pseudocode of the training and sampling process is presented in Appendix A.

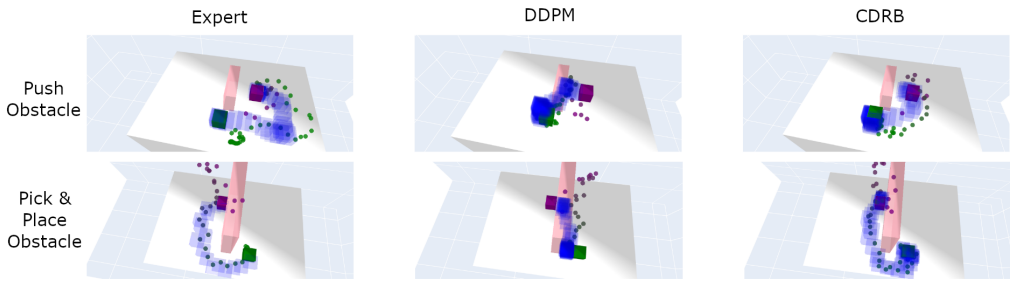


Figure 2: Trajectories on Pick & Place Obstacle and Push Obstacle. DDPM-generated trajectories frequently struggle with crashing into the wall. In contrast, Cold diffusion manages to learn how to avoid the wall constraints in the environment.

5 Experimental Evaluation

We conducted our experiments on a range of environments that includes various challenging obstacles that obstruct the agent's direct access to target objects or goal locations. We incorporated obstacles into existing environments to evaluate the agent's ability to navigate around obstacles while achieving its goals.

1) *Maze and Maze Tight:* These environments simulate a multi-goal maze with obstacles in the center. There are four possible goals located at the corners of the environment, and one goal location is randomly selected at each run. The Maze Tight variant is an adaptation of the original Maze, featuring narrower passages between the obstacles to test the robot's precision movement skills.
2) *Reach and Reach Obstacle:* In these environments, the primary objective is to navigate the robotic arm (gripper) to a specified target location. The Reach Obstacle variant introduces two tall walls on each side of the starting gripper position to test the robot's capability to reach the goal outside of the walls.
3) *Pick and Place and Pick and Place Obstacle:* These environments challenge the robot to grasp a block using the gripper and relocate it to a specified goal location. The obstacle version of this environment adds further complexity by including a tall wall that the robot arm must go up or around to reach the block or goal location.
4) *Push and Push Obstacle:* In these environments, the robot's objective is to push a block to a target location without activating the gripper fingers. The Push Obstacle environment includes a short wall placed in the center.

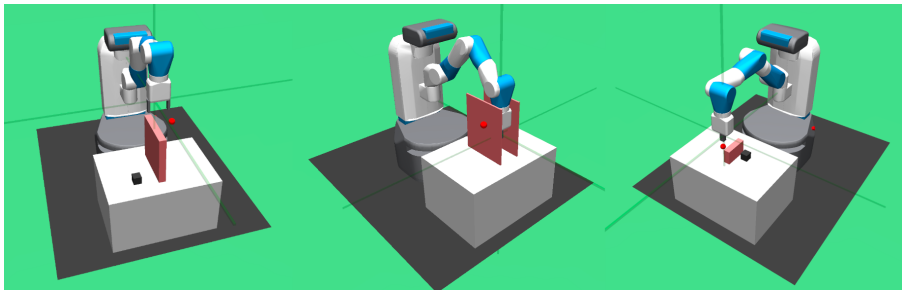


Figure 3: Environments considered in this paper. Left: Fetch robot must pick and place a box while avoiding a wall. Center: Robot starts between two walls and must reach a point outside the walls. Right: Robot must push a block to a desired point while avoiding an obstacle.

For each environment, we first generated expert data through the training of proficient agents via multi-goal RL algorithms. For the maze environments, multi-goal Soft Actor Critic (SAC) models

were used. For Reach, PickAndPlace, Push, and their obstacle variants, we utilized HER models to train the agent. We compare against [21, 1, 20], three recently proposed methods that leverage Gaussian diffusion for imitation learning.

Success Rate (%)	MLP	DDPM	Pearce, et al. 2023	Diffusion-QL	CDRB (ours)
Maze	74.22 ±3.81	100.0 ±0.00	90.13 ±1.21	100.0 ±0.00	95.33 ±2.05
Maze Tight	38.29 ±1.80	97.00 ±0.82	89.21 ±1.20	84.02 ±3.86	98.00 ±0.00
Reach Obstacle	80.74 ±5.20	89.27 ±0.61	89.63 ±0.41	74.03 ±8.12	89.61 ±1.31
Pick & Place Obstacle	0.00 ±0.00	58.53 ±4.17	58.80 ±2.34	24.19 ±8.31	70.94 ±4.65
Push Obstacle	2.43 ±1.10	39.27 ±13.93	34.81 ±7.79	29.41 ±5.24	66.67 ±1.21
Reach	90.34 ±2.18	100.0 ±0.00	94.02 ±1.03	58.60 ±14.69	100.0 ±0.00
Push	2.45 ±1.10	66.26 ±14.70	68.23 ±8.13	65.60 ±12.63	88.53 ±3.51
Pick & Place	0.00 ±0.00	91.47 ±7.51	61.00 ±3.56	23.60 ±9.63	94.73 ±3.78

Table 1: Benchmark results of test-time success rates goal reaching. Results over 5 seeds.

Through our experimental evaluation, we have identified two notable failure modes: 1) instances where the agent becomes stuck due to following planned trajectories that include states in the wall, and 2) challenges faced by the agent in precise planning to accomplish the desired goal. Figure 2 shows an example of trajectories generated by DDPM that directly cut through the wall and result in failures during execution time. In contrast, CDRB-generated trajectories only contain feasible states and thus would not cause the gripper to collide with the wall during execution.

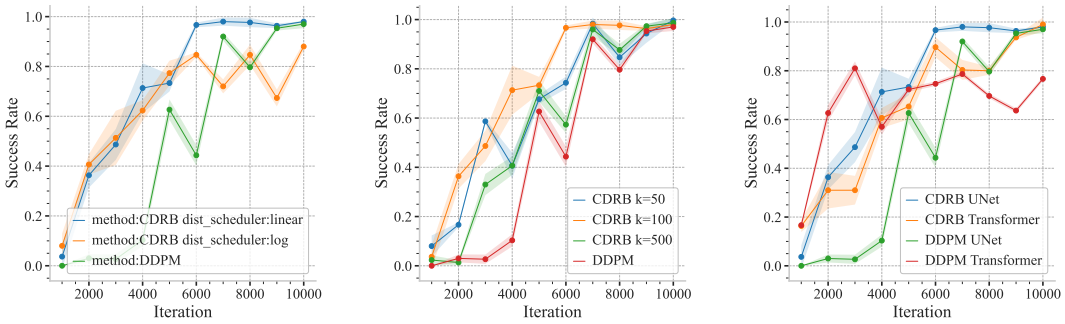
Specifically, in the Push Obstacle environment, approximately 60.73% of the trajectories generated by DDPM resulted in failure, with approximately 37.9% of these failures caused by trajectory states that are in the wall. The remaining 62.1% were due to difficulties pushing the block (e.g. failing to push the block all the way to the goal, or pushing the block off the table). Similarly, within the Pick and Place Obstacle environment, DDPM exhibited a failure rate of 41.5%, whereas 86.3% of these failures can be attributed to generated states that involve traversing through the wall. In contrast, 100% of CDRB's errors were of the second type, showing it learns to avoid the wall during training.

5.1 Further Discussion

K-Means Clustering: Keeping a full replay buffer can be costly, since diffusion across a larger buffer could have a higher propensity to make small adjustments at each reverse diffusion step. We thus experimented with applying K-Means clustering to reduce the buffer size. After clustering the buffer, we clip the resultant cluster centroids to the closest buffer points to ensure the feasibility of these states. The cluster centroids then become our new buffer. In general, we found this improved computational efficiency without degrading performance. See Figure 4 (b).

Temporal Transformers: Recent work has proposed the use of temporal transformers for diffusion-based imitation [20]. In contrast, the original Diffuser used a temporal U-Net [1]. Curious, we explored both architectures for learning $\mathcal{R}_\theta$. Looking at Figure 4 (c), we see that transformers generally converged much faster, but ultimately achieved worse final results. This is consistent with results in Table 1, where [20] often performed slightly worse than [1].

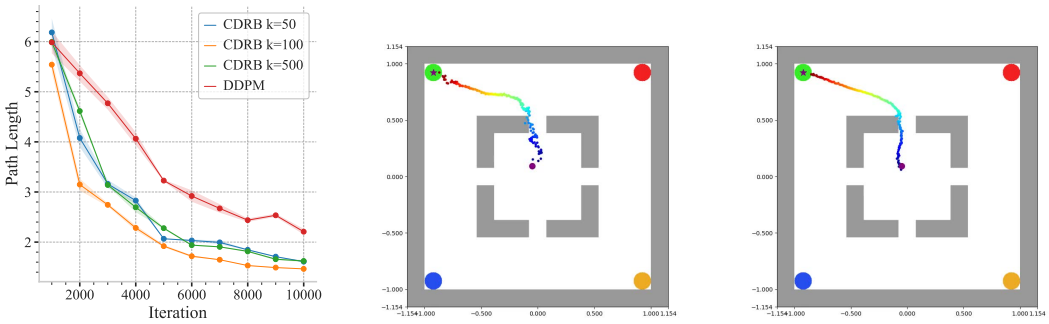
Distance Schedule: We wondered if change the rate of degradation in our forward diffusion process would impact learning (Equation 2). Using both a log and a linear distance schedule, we see the log schedule learns slightly faster, but is generally less performant at convergence. See Figure 4 (a). We suspect a slower derogation schedule causes our algorithm to spend too much time optimizing smaller regions of the state space.



(a) Distance Schedule (b) K-Means Size (c) Transformer vs U-Net

Figure 4: Examples of the predicted trajectory by CDRB and DDPM in Maze Tight environment

How smooth are the generated trajectories? During evaluation, we noticed that trajectories generated by CDRB appeared smoother than those generated by DDPM. See the example trajectories in Figure 5 (b) and (c). To test this, we divide the path length of the trajectories generated by DDPM and CDRB by the path length of a trajectory generated by SAC. We see that under this metric, CDRB generates more efficient paths to the goal. In particular, it seems much better at locking onto a smooth region at the start of its trajectory, as opposed to DDPM, which struggles at initial timesteps. This result is shown in Figure 5(a).



(a) Normalized Path Length (b) DDPM (c) CDRB

Figure 5: Examples of the predicted trajectory by CDRB and DDPM in Maze Tight environment

6 Limitations and Discussion

While we were generally impressed with the capabilities of CDBR, the algorithm has several bothersome limitations at this time. The algorithm seems sensitive to the number of reverse diffusion steps at test time. In fact, sensitivity in this regard is worse than most extant baselines. Because of our dependence on the replay buffer, the algorithm tends to struggle in areas of low visitation density, such as the Fly Trap environment that inspired much of our analysis [26].

Much like [1], we find that directly using diffusion-generated actions often works poorly, and had to often resort to inverse dynamics control. Yet, recent work has seen success with directly using diffusion for control [41], which suggests it should be possible. In contrast to prior work, [21, 41], we found our trained policies struggled in multi-modal environments. When initializing from human-provided demonstrations, rather than RL-demos, we found performance of all diffusion-based imitation learners suffered. This is a well-known problem in LfD [42]. We remain optimistic that future efforts in diffusion-based imitation will bear fruit in this area.

References

[1] M. Janner, Y. Du, J. B. Tenenbaum, and S. Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

[2] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3 (6):233–242, 1999.

[3] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.

[4] A. Y. Ng, S. Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[5] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

[6] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[7] J. Ho, J. Gupta, and S. Ermon. Model-free imitation learning with policy optimization. In *International Conference on Machine Learning*, pages 2760–2769. PMLR, 2016.

[8] J. Ho and S. Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[10] S. K. S. Ghasemipour, R. Zemel, and S. Gu. A divergence minimization perspective on imitation learning methods. In *Proceedings of the Conference on Robot Learning (CoRL)*, pages 1259–1277, 2020.

[11] X. Zhang, Y. Li, Z. Zhang, and Z.-L. Zhang. f-gail: Learning f-divergence for generative adversarial imitation learning. *Advances in neural information processing systems*, 33:12805–12815, 2020.

[12] I. Durugkar, M. Tec, S. Niekum, and P. Stone. Adversarial intrinsic motivation for reinforcement learning. *Advances in Neural Information Processing Systems*, 34:8622–8636, 2021.

[13] Y. J. Ma, J. Yan, D. Jayaraman, and O. Bastani. How far i'll go: Offline goal-conditioned reinforcement learning via f-advantage regression. *arXiv preprint arXiv:2206.03023*, 2022.

[14] S. Reddy, A. D. Dragan, and S. Levine. Sqil: Imitation learning via reinforcement learning with sparse rewards. *arXiv preprint arXiv:1905.11108*, 2019.

[15] Y. Ding, C. Florensa, P. Abbeel, and M. Phielipp. Goal-conditioned imitation learning. In *Advances in Neural Information Processing Systems*, pages 15298–15309, 2019.

[16] K. Menda, K. Driggs-Campbell, and M. J. Kochenderfer. Ensembledagger: A Bayesian approach to safe imitation learning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5041–5048, 2019.

[17] H. Yin, P. Seiler, M. Jin, and M. Arcak. Imitation learning with stability and safety guarantees. *IEEE Control Systems Letters*, 6:409–414, 2021.

[18] J. Chen, B. Yuan, and M. Tomizuka. Deep imitation learning for autonomous driving in generic urban scenarios with enhanced safety. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2884–2890, 2019.

[19] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.

[20] T. Pearce, T. Rashid, A. Kanervisto, D. Bignell, M. Sun, R. Georgescu, S. V. Macua, S. Z. Tan, I. Momennejad, K. Hofmann, and S. Devlin. Imitating human behaviour with diffusion models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

[21] Z. Wang, J. J. Hunt, and M. Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*, 2022.

[22] E. Altman. *Constrained Markov decision processes*, volume 7. CRC press, 1999.

[23] A. Hans, D. Schneegaß, A. M. Schäfer, and S. Udluft. Safe exploration for reinforcement learning. In *ESANN*, pages 143–148. Citeseer, 2008.

[24] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel. Motion planning with sequential convex optimization and convex collision checking. *International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[25] S. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[26] A. Yershova, L. Jaillet, T. Siméon, and S. M. LaValle. Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3856–3861, Barcelona, April 2005.

[27] J. F. Fisac, N. F. Lugovoy, V. Rubies-Royo, S. Ghosh, and C. J. Tomlin. Bridging Hamilton-Jacobi safety analysis and reinforcement learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 8550–8556, 2019.

[28] C. Gehring and D. Precup. Smart exploration in reinforcement learning using absolute temporal difference errors. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1037–1044, 2013.

[29] T. W. Bickmore, H. Trinh, S. Olafsson, T. K. O'Leary, R. Asadi, N. M. Rickles, and R. Cruz. Patient and consumer safety risks when using conversational assistants for medical information: an observational study of siri, alexa, and google assistant. *Journal of medical Internet research*, 20(9):e11510, 2018.

[30] T. M. Moldovan and P. Abbeel. Safe exploration in markov decision processes. *arXiv preprint arXiv:1205.4810*, 2012.

[31] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[32] B. C. Stadie, P. Abbeel, and I. Sutskever. Third-person imitation learning. *arXiv preprint arXiv:1703.01703*, 2017.

[33] D. Hadfield-Menell, S. J. Russell, P. Abbeel, and A. Dragan. Cooperative inverse reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[34] J. Leike, D. Krueger, T. Everitt, M. Martic, V. Maini, and S. Legg. Scalable agent alignment via reward modeling: a research direction. *arXiv preprint arXiv:1811.07871*, 2018.

[35] M. Pecka and T. Svoboda. Safe exploration techniques for reinforcement learning–an overview. In *Modelling and Simulation for Autonomous Systems: First International Workshop, MESAS 2014, Rome, Italy, May 5-6, 2014, Revised Selected Papers 1*, pages 357–375. Springer, 2014.

[36] J. Garcıa and F. Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[37] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6840–6851, 2020.

[38] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[39] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[40] A. Bansal, E. Borgnia, H.-M. Chu, J. S. Li, H. Kazemi, F. Huang, M. Goldblum, J. Geiping, and T. Goldstein. Cold diffusion: Inverting arbitrary image transforms without noise. *arXiv preprint arXiv:2208.09392*, 2022.

[41] T. Yoneda, L. Sun, B. Stadie, G. Yang, and M. Walter. To the noise and back: Diffusion for shared autonomy. *arXiv preprint arXiv:2302.12244*, 2023. doi:10.48550/ARXIV.2302.12244. URL https://arxiv.org/abs/2302.12244.

[42] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *arXiv preprint arXiv:2108.03298*, 2021.

[43] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

A Algorithm Pseudocode

Algorithm 1 Training the Noise Model

Require: Expert trajectories $\mathcal{T}$, distance schedule ϵ, number of diffusion steps t, replay buffer $\mathcal{B}$
Output: Trained restoration operator $\mathcal{R}_\theta$

```
repeat
    τ ~ T   (Sample a trajectory from expert trajectories)
    k ~ ({1, ..., t − 1})   (Sample a diffusion step uniformly)
    for i = 0, ..., T − 1
        s_i(k) ~ B_{ε_k}(s_i) ∩ B   (Sample a RB point within a ball around s_i with radius ε_t)
    end for
    Construct noisy trajectory τ(k) = [s_0(k), s_1(k), ..., s_T(k), s_T(k)]
    τ(0) = R_θ(τ(k), k)
    Perform gradient descent on min_θ E_{τ~ρ(τ)} ‖τ(0) − τ‖
until converged
```

Algorithm 2 Sampling Process: Generating new Trajectories

Input: Replay buffer $\mathcal{B}$, start state s_0, goal state s_T
Required: Pretrained restoration operator $\mathcal{R}_\theta$, pretrained controller f_ϕ, number of diffusion timesteps t
Output: State-action sequence $\tau(0) = [s_0, a_0, \ldots, s_T, a_T]$

```
Sample T + 1 state-action pairs τ(t) uniformly from B
for k = t, t − 1 ..., 1 do
    τ̂(0) = R_θ(τ(k); k)
    τ(k − 1) = D(τ̂(0); k − 1)
end for
for i = 0, ..., T − 1
    Compute the action a_i = f_φ(s_i, s_i + 1)
end for
```

B Experimental Details

In both the Maze and Maze Tight environments, we initially trained expert policies using the Soft Actor Critic (SAC) method for 1,000,000 training timesteps, utilizing sparse reward. [43]. Then, we collect the 8k demonstrations as the training dataset by using SAC, and train the diffusion models with this data. Since the length of the demonstration trajectories is different, we randomly crop the demonstration trajectories so that the trajectory length becomes equal for training. The cropped length is 168 in this experiment. In the inference stage, we randomly sample starts and goals while ensuring that the goal can be reached within 168 steps. The goals are sampled from 4 points in the center of the colored circle near the corner in Fig. 1. The starts are sampled from the inner region surrounded by 4 walls located around the center. Then, the diffusion model plans the trajectory by inpainting manner in the same way as diffusion planning[1]. The action is obtained by a PD controller, which calculates the torque so that the agent follows the trajectory predicted by the diffusion model. If the agent touches the goal circle, the planning is regarded as a success.

In the Fetch environments, we obtained an expert policy with Hindsight Experience Replay, trained for two million training steps. These environments were all multi-goal with sparse reward binary

feedback. Goals are sampled uniformly from the environment, and involve reaching a target location or moving a block to the target location. We collected 5,000 demonstrations for each environment, each of length 60. It should be noted that on Fetch Pick Obstacle and Fetch Push Obstacle, the best HER controller we could train received only a 90% success rate, suggesting this environment is difficult even for RL agents. Unlike the maze environments, no validation set is used. Instead, the average success is measured by randomly resetting the environment K-times and running the diffusion policy forward. Actions are obtained with a PD controller, which allows the agent to follow the paths generated by the diffusion model. If the PD controller executing the plan manages to get within ϵ of the target goal at the last timestep, the trajectory is counted as a success.

C Some Additional Analysis

In this section, we try to understand the difference in average transport costs paid by DDPM vs cold diffusion over a replay buffer. As we will see, the total mass that must be moved is less when operating over a replay buffer, which suggests that perhaps CDRB can converge in fewer diffusion steps overall. We see this bear out empirically, with cold diffusion generally converging faster.

Recall the Wasserstein Metric W_1 can be defined for probability distributions via

$$W_1(P, Q) = \inf_{\gamma \in \Pi(P,Q)} \mathbb{E}_{(x,y)\sim\gamma} \left[d(x, y) \right]$$

Where $\Pi(P, Q)$ is the set of all joint probability distributions with marginals P, Q. It is common to instead use Kantorovich-Rubinstein duality to write this condition as

$$W_1(P, Q) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim P} \left[f(x) \right] - \mathbb{E}_{y\sim Q} \left[f(y) \right]$$

With Suprememum taken over the class of ℓ^1-Lipschitz continuous functions.

Turning our eyes towards the transport cost of diffusion, Let $\rho(\tau(t))$ be the prior distribution on the diffusion process, i.e. $\rho(\tau(t)) \sim \mathcal{N}(\mu, \sigma)$ Then we have the following computational proposition.

Proposition 1. *Assume we are on some line ℓ with width d. Let $\rho(\tau(t)) \sim \mathcal{N}(\mu, \sigma)$. If we assume the buffer $\mathcal{B}$ is Uniformly distributed across ℓ then we have*

$$W_1(\rho(\tau_{\mathcal{B}}), \rho(\tau)) = W_1(\rho(\tau(t)), \rho(\tau(0)) \cdot \frac{|\mathcal{B}|(1-\epsilon)}{d^2}$$

Where ϵ is the width of the search radius in the CDRB graph search.

This calculation essentially says that we can expect to pay some extra cost for the CDRB graph search procedure, both in terms of the buffer size $|\mathcal{B}|$ and the search radius ϵ. However, in turn we get more efficient movement through the state space ℓ, thanks to the factor of d^2. In many situations, we would expect this lower transport cost to result in convergence of CDRB with fewer diffusion steps. We see this borne out in our experimental results.

When there is a hole in the feasible space, this result changes slightly to account for the extra cost associated with moving across the infeasible region.

Proposition 2. *Under the assumptions of Proposition 1, if we have a hold in the feasible space of size r, then we see*

$$W_1(\rho(\tau_{\mathcal{B}}), \rho(\tau)) = W_1(\rho(\tau(t)), \rho(\tau(0)) \cdot \frac{|\mathcal{B}|(1-\epsilon)}{d^2 \cdot r}$$

Where ϵ is the width of the search radius in the CDRB graph search.