To the Noise and Back: Diffusion for Shared Autonomy

Author Names Omitted for Anonymous Review. Paper-ID [168]

***Abstract*—Shared autonomy is an operational concept in which a user and an autonomous agent collaboratively control a robotic system. It provides a number of advantages over the extremes of full-teleoperation and full-autonomy in many settings. Traditional approaches to shared autonomy rely on knowledge of the environment dynamics, a discrete space of user goals that is known a priori, or knowledge of the user's policy—assumptions that are unrealistic in many domains. Recent works relax these assumptions by formulating shared autonomy with model-free deep reinforcement learning (RL), and train the agent's policy that produces an action close to that of the user while satisfying value function constraints. These formulations inherently rely on human-in-the-loop training to learn the assistant's policy and, in practice, replace the user with a surrogate policy for the sake of training efficiency. In effect, this trades one difficulty for another. While we no longer need knowledge of the goal space (e.g., that the goals are discrete or constrained), we do need knowledge of a task-specific reward function to train the policy. Unfortunately, such reward specification can be a difficult and brittle process. In this paper, we present a new approach to shared autonomy that employs a modulation of the forward and reverse diffusion process of diffusion models. Our approach does not assume known environment dynamics or the space of user goals, and in contrast to previous work, it does not require any reward feedback, nor does it require access to the user's policy (surrogate or otherwise) during training. Instead, our framework learns a distribution over a potentially multimodal space of desired behaviors. It then employs a diffusion model to adapt the user's actions to a sample from this distribution. Crucially, we show that it is possible to carry out this process in a manner that preserves the user's control authority. We evaluate our framework on a series of challenging continuous control tasks, and analyze its ability to effectively correct user actions while maintaining their autonomy.**

I. Introduction

Contemporary robots primarily operate in one of two different ways—full teleoperation or full autonomy. Teleoperation is common in unstructured environments (e.g., underwater), where the proficiency with which robots are able to understand their surroundings is insufficient for fully autonomous robots to operate reliably. However, direct teleoperation requires users to interpret the robot's observations of the environment while simultaneously controlling its low-level actions, a responsibility that is particularly challenging for highly dynamic tasks. This operational gap motivates a setting in which a human and an autonomous agent collaborate and share the control of a robotic system.

Shared autonomy [2] is a framework in which a human user (also referred to as the *pilot*) performs a task with an assistance of an autonomous agent (also referred to as the *copilot*) [21, 41, 3, 13, 14, 22]. The role of the agent is to complement the control authority of the user, whether to improve the robot's performance on the current task or to encourage/ensure safe behavior. An important consideration when providing assistance via shared autonomy is the degree to which the agent balances the user's preference for maintaining control authority (i.e., the *fidelity* of the assisted behavior relative to the user's actions), and the potential benefits of endowing more control to the agent (i.e., the *conformity* of the assisted behavior to that of an autonomous agent).

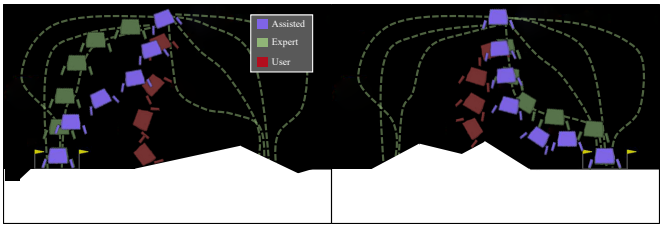


Fig. 1: Our framework utilizes a diffusion model to adapt a user's action (red) to those generated from an expert distribution (green) in a manner (blue) that balances a user's desire to maintain control authority with the benefits (e.g., safety) of conforming to the expert distribution learned from demonstrations. Without knowledge of the user's specific goal (e.g., the landing location), the expert distribution reflects different goals that one or more experts previously reached.

A core difficulty of shared autonomy lies in the fact that the user's goal (intent) is typically not known. Many approaches to shared autonomy assume that there is a fixed, discrete set of candidate goals and seek to infer the user's specific goal at test-time based on observations, including the user's control input [35, 29, 37, 24, 14]. Such assumptions may be reasonable in structured environments (e.g., in the context of a manipulation task when there is a small number of graspable objects sitting on a table). However, they can be limiting in unstructured environments that lack well-defined goals or that have a very large set of potential goals.

Bootstrapped by function approximation with neural-networks, recent deep reinforcement learning (RL) algorithms seek to learn assistive policies without assumptions on the nature or knowledge of the goal space, or the assumption that the environment dynamics are known. Reddy et al. [39] propose a deep RL approach to shared autonomy for domains with discrete actions, nominally relying on reward feedback from the user as an alternative to assuming that the goal space is known. In an effort to balance the user's control authority

with task performance, the assistant chooses the action most similar to that of the user while also satisfying a state-action value constraint. Schaff and Walter [43] treat the copilot as providing a residual that is added to the user's actions to correct for unsafe behavior. They train their model to minimize the norm of the residual, subject to a goal-agnostic reward constraint that represents safe behavior.

These methods treat the pilot as a part of the environment, using an augmented state that includes the user's action. Framing the problem in this way has a clear and significant advantage—it enables the direct utilization of the the modern suite of tools for deep RL. However, these methods have two notable limitations. First, they nominally require human-in-the-loop interaction during training in order to generate user actions while learning the assistant's policy. Since the sample complexity of deep RL makes this interaction intractable, these methods replace the human with a *surrogate policy*. If this surrogate is misspecified or invalid, this approach can lead to copilots that are not well-suited to work with actual human pilots [43]. Second, these methods require access to a task-specific reward function during training, which may be difficult to obtain in practice.

In light of these limitations, we propose a model-free approach to shared autonomy that interpolates between the user's action and an action sampled from a generative model that provides a distribution over desired behavior (Fig. 1). Our approach has the distinct advantage that it does not require knowledge of or access to the user's policy or any reward feedback during training and, in turn, no reward engineering. Instead, training – which involves learning the generative model – only requires access to trajectories that are representative of desired behavior.

The generative model that underlies our approach is a diffusion model [49, 45, 46, 25], which has proven highly effective for complex generation tasks including image synthesis [12, 42]. Diffusion models consist of two key processes: the *forward process* and the *reverse process*. The forward process iteratively adds Gaussian noise to the input with an increasing noise scale, while the reverse process is trained to iteratively denoise a noisy input in order to arrive at the target distribution. As part of this denoising process, the model produces a gradient that shows a direction to which the likelihood of its input increases under the target distribution. Once the model is trained, generating a sample from the (unknown) target distribution amounts to running the reverse process on a sample drawn from a zero-mean isotropic Gaussian distribution.

As we will see in the following sections, a direct use of diffusion models for shared autonomy ends up in generating an action that ignores user's intent (i.e., low *fidelity* to user intent), even though the action would be consistent with the desired behaviors (i.e., high *conformity* to the target behaviors). To address this, we propose a new algorithm that controls the effect of the forward and reverse process through a *forward diffusion ratio* γ (Fig. 2), that regulates the balance between the fidelity and the conformity of the generated actions. The forward diffusion ratio provides a formal bound on the extent to which the copilot's action deviates from that of the user.

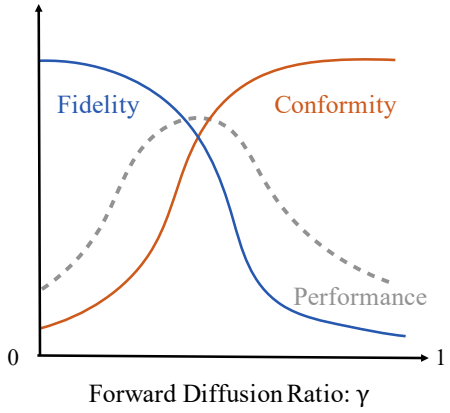


Fig. 2: Our algorithm uses the *forward diffusion ratio* γ to regulate the extent of forward and reverse diffusion. When γ is small, the assistant generates actions nearly identical to those of the user (i.e., high fidelity) but may be far from the desired behaviors (i.e., low conformity). Depending on the competency of the user, these actions may result in poor task performance (e.g., due to crashes). As with standard diffusion, when γ is large, the assistant will generate actions that have high likelihood under the distribution over desired behavior (i.e., high conformity), with little-to-no regard for their similarity to the user's actions (i.e., low fidelity). While these actions will be safe, they will likely result in poor performance since the forward process has removed any information about user's intention. By controlling γ, our algorithm identifies a regime that trades off between fidelity and conformity to generate actions that preserve the user's intent and, in turn, the critical information to complete the task.

We evaluate our shared autonomy algorithm using a series of continuous control tasks. In each case, we demonstrate that our algorithm significantly improves the performance of a variety of different pilots, and we analyze the effects of a range of different diffusion ratios. Empirically, we find that there exists a consistent setting of the diffusion ratio that generalizes across a variety of different pilot policies, and that ratios below and above this setting have expected effects on the fidelity and conformity of the resulting actions.

II. Method

Integral to our approach to shared autonomy is its utilization of diffusion models as a generative model that serves to correct the actions of the user. As such, we begin this section with an in-depth review of diffusion models.

A. Background on diffusion models

We first present the mathematical formulation of diffusion models, with a particular emphasis on the denoising diffusion probabilistic model (DDPM) [25], which we employ.

A probabilistic diffusion model [45] is a type of generative model that is characterized by two processes (Fig. 3): a

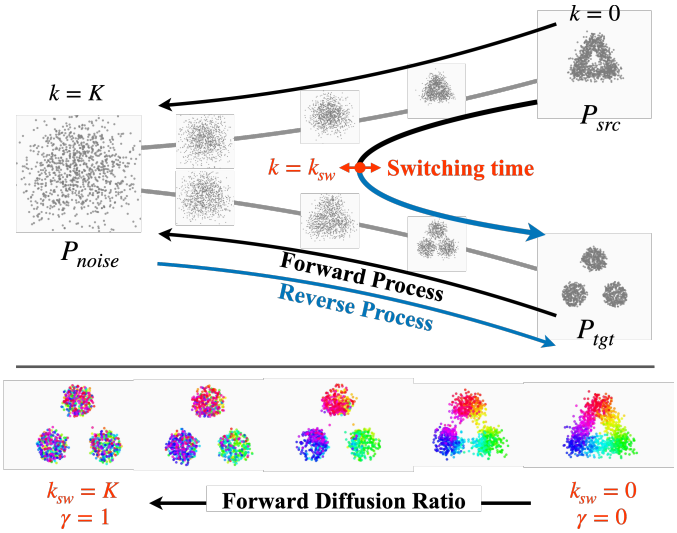


Fig. 3: (Top) A visualization of the forward diffusion of a source distribution P_{src} and its reverse diffusion to the target distribution P_{tgt}. (Bottom) The result of forward and reverse diffusion for different switching times k_{sw}, where standard reverse diffusion process corresponds to $k_{\text{sw}} = K$.

forward diffusion process and a *reverse diffusion process*. The forward diffusion process is an iterative first-order Markov chain that adds noise to a sample from a data distribution $\boldsymbol{x}_0 \sim q(\boldsymbol{x}_0)$. Assuming a predefined sequential noise schedule of K steps, $\beta_1, \ldots, \beta_K$ and $\alpha_k := 1 - \beta_k$, a single step in the forward process operates as

$$\boldsymbol{x}_k = \sqrt{\alpha_k}\boldsymbol{x}_{k-1} + \sqrt{1-\alpha_k}\boldsymbol{\epsilon}, \;\; \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}). \tag{1}$$

Applying this recursively, multiple steps of forward process can be written in a closed form as

$$\boldsymbol{x}_k = \sqrt{\bar{\alpha}_k}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_k}\boldsymbol{\epsilon}, \;\; \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), \tag{2}$$

where $\bar{\alpha}_k := \Pi_{s=1}^{k} \alpha_s$. Equivalently,

$$q(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}) = \mathcal{N}(\boldsymbol{x}_k; \sqrt{\alpha_k}\boldsymbol{x}_0, (1-\alpha_k)\boldsymbol{I}) \tag{3a}$$
$$q(\boldsymbol{x}_k \mid \boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_k; \sqrt{\bar{\alpha}_k}\boldsymbol{x}_0, (1-\bar{\alpha}_k)\boldsymbol{I}) \tag{3b}$$

Meanwhile, the *reverse diffusion process* is a Markov chain that iteratively denoises a noisy input, starting from $\boldsymbol{x}_K \sim N(\mathbf{0}, \boldsymbol{I})$. The conditional distribution over the output at each step $q(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k)$ is Gaussian. However, computing these intermediate steps depends upon the entire data distribution and is generally not tractable in closed-form. Thus, we train a model that approximates it as $p_\theta(\boldsymbol{x}_{k-1} \mid \boldsymbol{x}_k) = \mathcal{N}(\boldsymbol{x}_{k-1}; \boldsymbol{\mu}_\theta(\boldsymbol{x}_k, k), \sigma_k^2\boldsymbol{I})$ to run the reverse process. Happily, the probability $q(\boldsymbol{x}_{k-1} \mid \boldsymbol{x}_k)$ that we want to model becomes tractable if additionally conditioned on $\boldsymbol{x}_0$. By Bayes' rule, we have

$$q(\boldsymbol{x}_{k-1} \mid \boldsymbol{x}_k, \boldsymbol{x}_0) = q(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}, \boldsymbol{x}_0)\frac{q(\boldsymbol{x}_{k-1} \mid \boldsymbol{x}_0)}{q(\boldsymbol{x}_k \mid \boldsymbol{x}_0)} \tag{4a}$$
$$= q(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1})\frac{q(\boldsymbol{x}_{k-1} \mid \boldsymbol{x}_0)}{q(\boldsymbol{x}_k \mid \boldsymbol{x}_0)} \tag{4b}$$
$$= \mathcal{N}(\boldsymbol{x}_{k-1}; \tilde{\boldsymbol{\mu}}(\boldsymbol{x}_k, \boldsymbol{x}_0), \tilde{\beta}_k\boldsymbol{I}) \tag{4c}$$

Crucially, Equation 4b only contains known terms from the Gaussian distributions in Equations 3a and 3b. In the last expression, we have dropped the mean and covariance for notational brevity.

Intuitively, we can train the model $p_\theta(\boldsymbol{x}_{k-1} \mid \boldsymbol{x}_k)$ by minimizing its divergence from $q(\boldsymbol{x}_{k-1} \mid \boldsymbol{x}_k, \boldsymbol{x}_0)$ over samples from the data distribution $q(\boldsymbol{x}_0)$. Utilizing the evidence lower bound of $\mathbb{E}_q[-\log p_\theta(\boldsymbol{x}_0)]$, we can formulate the loss as:

$$\begin{aligned} \mathcal{L} &:= \mathbb{E}_q\left[\sum_{k=2}^{K-1} L_k - \log p_\theta(\boldsymbol{x}_0 \mid \boldsymbol{x}_1) + C\right], \\ L_k &= D_{\text{KL}}\big(q(\boldsymbol{x}_{k-1} \mid \boldsymbol{x}_k, \boldsymbol{x}_0) \| p_\theta(\boldsymbol{x}_{k-1} \mid \boldsymbol{x}_k)\big), \end{aligned} \tag{5}$$

where D_{KL} is the Kullback-Leibler (KL) divergence, C is a constant that corresponds to fixed parameters of the forward diffusion process [25]. The term L_k is the KL divergence between two Gaussian distributions, which can be computed in closed form as the squared error between two means

$$L_k = \mathbb{E}_{\boldsymbol{x}_k \sim q}\left[\frac{1}{2\sigma_k^2}\|\tilde{\boldsymbol{\mu}}(\boldsymbol{x}_k, \boldsymbol{x}_0) - \boldsymbol{\mu}_\theta(\boldsymbol{x}_k, k)\|_2^2\right] + C', \tag{6}$$

where C' is a constant independent of θ. Now we note the fact that predicting $\hat{\boldsymbol{x}}_{k-1} = \boldsymbol{\mu}_\theta(\boldsymbol{x}_k, k)$ given $\boldsymbol{x}_k$ is equivalent to predicting the noise $\boldsymbol{\epsilon}$ that was added to $\boldsymbol{x}_{k-1}$ (see Equation 1). We can thus reformulate the problem as one of minimizing the error in the noise prediction. By parameterizing $\boldsymbol{\mu}_\theta(\boldsymbol{x}_k, k)$ as a function of predicted noise $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_k, k)$, we can simplify Equation 6 to

$$\mathbb{E}_{q, \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})}\left[c_k(\beta_{1:K}, \sigma_k)\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_k, k)\|_2^2\right], \tag{7}$$

where $c_k(\beta_{1:K}, \sigma_k)$ is a constant computed from the beta schedule for each timestep k. Ho et al. [25] further simplify this equation to arrive at the final loss that, with a slight abuse of notation, becomes

$$\mathcal{L}_{\text{simple}} := \mathbb{E}_{k, \boldsymbol{x}_0, \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})}\left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_k(\boldsymbol{x}_0, \boldsymbol{\epsilon}), k)\|_2^2\right] \tag{8a}$$
$$\boldsymbol{x}_k(\boldsymbol{x}_0, \boldsymbol{\epsilon}) := \sqrt{\bar{\alpha}_k}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_k}\boldsymbol{\epsilon}, \tag{8b}$$

where k is sampled uniformly as $k \in [1, K]$. After training, we can generate a sample $\boldsymbol{x}_0$ by running the reverse diffusion process recursively from $\boldsymbol{x}_K \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

$$\boldsymbol{x}_{k-1} = \mu_{\theta^*}(\boldsymbol{x}_k, k) + \sigma_k \boldsymbol{z}, \tag{9}$$

where $z \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ for $k > 1$, else $\boldsymbol{z} = \mathbf{0}$.

B. Diffusion model guidance for distribution transformation

Consider generally the problem of transforming one distribution P_{src} onto another distribution P_{tgt}. This problem is challenging, because finding such a correspondence between distributions has an inherent dependence on the intrinsic geometrical match between the two distributions. However, there is an enticing shortcut that presents itself in the form of diffusion processes. In particular, we note that the forward diffusion process should allow us to map P_{src} into a noise distribution P_{noise} (i.e., zero-mean isotropic Gaussian noise). Drawing samples from this noise distribution, we can execute the reverse diffusion process to guide the sample to being drawn from P_{tgt}, assuming of course that the reverse process was trained on data samples from P_{tgt}.

Although this diffusion process will ultimately take points $\boldsymbol{x}_{\text{src}}$ from P_{src} and map to samples drawn from P_{tgt}, resulting in $\hat{\boldsymbol{x}}_{\text{tgt}}$, it will do so in a destructive manner. In particular, as the forward diffusion process is applied to $\boldsymbol{x}_{\text{src}}$, the distribution of these points becomes progressively more random (by design). In the end, its distribution reaches an isotropic Gaussian distribution, completely corrupting the information in the original sample. At this point, there is no relationship between the source points $\boldsymbol{x}_{\text{src}}$ and the derived points $\hat{\boldsymbol{x}}_{\text{tgt}}$. We may as well have simply started from Gaussian noise and run reverse diffusion onto P_{tgt}.

We would like to learn a transformation $\mathcal{F} : \boldsymbol{x}_{\text{src}} \rightarrow \hat{\boldsymbol{x}}_{\text{tgt}}$ that preserves information. Such a transformation between distributions can be achieved by running the forward diffusion process on $P_{\text{src}}(x)$ *partway through*, up to $k = k_{\text{sw}}$, followed by the reverse diffusion process for the same number of steps (k_{sw}), in order to transform these partially diffused points onto P_{tgt}. Crucially, this process trades off the amount of information present in $\boldsymbol{x}_{\text{src}}$ that is preserved as a result of forward diffusion, and the consistency (in terms of likelihood) of the sample generated via reverse diffusion with respect to the target distribution P_{tgt} (i.e., the distribution over desired behaviors).

Very recently, Meng et al. [33] proposed a similar partial diffusion process for the task of generating realistic images based upon a user-provided sketch. The authors derive a bound on the distance between $\hat{\boldsymbol{x}}_{\text{tgt}}$ and $\boldsymbol{x}_{\text{src}}$ as follows.[1] Assuming that $\|\boldsymbol{\epsilon}_\theta(\boldsymbol{x}, k)\| \leq \mathcal{K}$ for all $\boldsymbol{x} \in X$ and $k \in [0, K]$, then for all $\delta \in (0, 1)$ with probability at least $(1 - \delta)$,

$$\|\boldsymbol{x}_{\text{src}} - \hat{\boldsymbol{x}}_{\text{tgt}}\|_2^2 \leq \sigma_{k_{\text{sw}}}^2 (\mathcal{K}\sigma_{k_{\text{sw}}}^2 + d + 2\sqrt{-d \cdot \log \delta} - 2 \log \delta), \tag{10}$$

where d is the dimensionality of $\boldsymbol{x}$. From this bound, we can see that the distance between $\boldsymbol{x}_{\text{src}}$ and $\hat{\boldsymbol{x}}_{\text{tgt}}$ increases with k_{sw}, since $\sigma_{k_{\text{sw}}}$ increases with k_{sw} while other terms are not affected. This expression allows us to bound the difference between the action generated by the assistant $\hat{\boldsymbol{x}}_{\text{tgt}}$ and that of the user $\boldsymbol{x}_{\text{src}}$.

[1]Their bound is for a variance exploding (VE) formulation of diffusion, whereas we employ DDPM that uses a variance preserving (VP) formulation. Despite the difference, they share the same mathematical intuition.

Algorithm 1 Shared autonomy as partial diffusion

Input: Observation $\boldsymbol{s}_t$ and pilot's action $\boldsymbol{a}_t^h$
Output: Shared action $\boldsymbol{a}_t^s$
Require: A pretrained state-conditioned denoising model $\mu_{\theta^*}(\boldsymbol{a}, k \mid \boldsymbol{s})$, forward diffusion ratio γ, diffusion timestep K

Compute the switching timestep $k_{\text{sw}} \leftarrow \text{ToInteger}(\gamma K)$
Sample a Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
$\tilde{\boldsymbol{a}}_{t,k_{\text{sw}}}^h \leftarrow \sqrt{\bar{\alpha}_{k_{\text{sw}}}}\boldsymbol{a}_t^h + \sqrt{1 - \bar{\alpha}_{k_{\text{sw}}}}\boldsymbol{\epsilon}$ ▷ Forward process
$\boldsymbol{a}_{t,k_{\text{sw}}}^s \leftarrow \tilde{\boldsymbol{a}}_{t,k_{\text{sw}}}^h$
for k in $k_{\text{sw}}, \ldots, 2$ **do**
  Sample a noise vector $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
  $\boldsymbol{a}_{t,k-1}^s \leftarrow \boldsymbol{\mu}_{\theta^*}(\boldsymbol{a}_{t,k}^s, k \mid \boldsymbol{s}_t) + \sigma_k \mathbf{z}$ ▷ Reverse process
end for
$\boldsymbol{a}_{t,0}^s \leftarrow \boldsymbol{\mu}_\theta(\boldsymbol{a}_{t,1}^s, 1 \mid \boldsymbol{s}_t)$
return $\boldsymbol{a}_{t,0}^s$

Figure 3 visualizes this property in two dimensions. Here, samples from P_{src} form the shape of a triangle, and samples from P_{tgt} follow a distribution with three modes, each centered on one of the vertices of a triangle. The black and blue arrow represents the idea of executing partial diffusion—running the forward (black) and reverse (blue) processes for k_{sw} steps. At the bottom of Figure 3, we visualize the result of partial forward and reverse diffusion for different values of k_{sw}. Here, we color points based on their original spatial location in P_{src} (e.g., green points in the lower-right, red at the top, and blue at the lower-left), and track their location over different degrees of forward and reverse diffusion. Based on the visualization of the resulting distributions, we see that the distance between the initial points $\boldsymbol{x}_{\text{src}}$ and $\hat{\boldsymbol{x}}_{\text{tgt}}$ increases with k_{sw}, consistent with Equation 10. More generally, we find that:

When k_{sw} is small:

- The original information is well-preserved (i.e., small displacements; high *fidelity*)
- The obtained distribution is far from P_{tgt} (i.e., low *conformity*)

When k_{sw} is large:

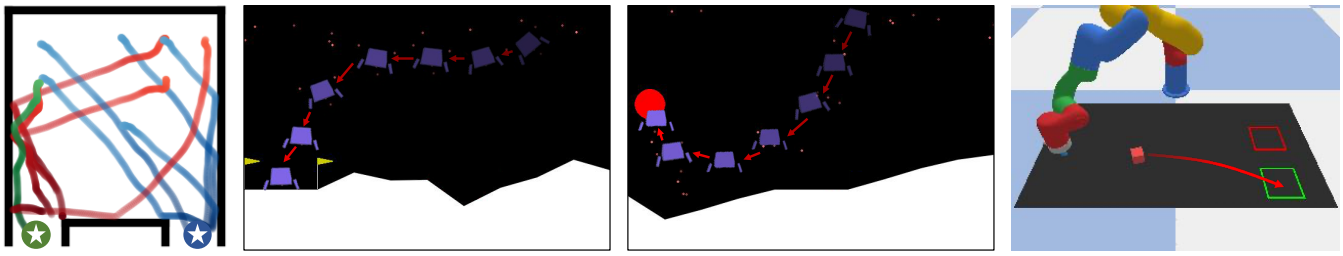
- The original information is corrupted (i.e., large displacements; low *fidelity*)
- The obtained distribution is close to P_{tgt} (i.e., high *conformity*)

To discuss the effect of k_{sw} independent of the number of diffusion steps K, we herein define *Forward Diffusion Ratio* $\gamma := k_{\text{sw}}/K$, and will refer to this throughout the paper.

A similar idea in diffusion models has been adopted in image generation, manipulation or 3D geometry generation tasks [33, 50, 38, 52].

C. Distribution transformation for shared autonomy

In a shared autonomy task, a copilot is asked to produce a shared action a_t^s, given the current state s_t and pilot action a_t^h. Ideally, the copilot will intervene such that the corrected action a_t^s preserves the pilot's "intention". However, it is hard to formulate the pilot's "intention" in a well-defined way, particularly when the space of goals (tasks) is not known nor even well-defined.



(a) 2D-control (b) Lunar Lander (c) Lunar Reacher (d) Block Pushing

Fig. 4: We evaluate our algorithm in the context of four shared autonomy environments including a (a) 2D control task in which an agent navigates to one of two different goals, (b) Lunar Lander that tasks a drone with landing at a designated location, (c) a Lunar Reacher variant in which the objective is to reach a designated region in the environment, and (d) Block Pushing, in which the objective is to use a robot arm to push an object into one of two different goal regions.

Our algorithm assumes that the latent intent of the user has non-zero likelihood under the distribution over target behaviors that we learn to sample from using a diffusion model. In other words, if we have a finite set of expert demonstrations to train the diffusion model, the states or state-action pairs that a user would want to achieve need to exist in the demonstrations. With our formulation to shared autonomy, the copilot manages the trade-off between respecting pilot's action (i.e., *fidelity* to the pilot) and executing an action that is likely under the learned behavior distribution (i.e., *conformity* to the target behaviors). This trade-off bears similarity to the trade-off discussed in Section II-B based on Figure 3.

Now, given a state s_t, we can see P_{src} in Figure 3 as a distribution over pilot actions and P_{tgt} as a distribution over target behaviors, where each point represents a single action. For the sake of explanation, we can pretend that each cluster in P_{tgt} corresponds to a set of actions, each of which causes distinct transitions. Hence, each cluster can be thought of as a pilot's "intention", and the actions outside of the cluster as being undesirable. Considering the visualization of the different partially forward and reverse diffused distributions at the bottom of the figure, we see that as we increase γ from 0.0 to 1.0, most of the pilot's actions begin to go inside of the set of ideal actions. This captures the notion that conformity to the target behavior increases with γ. On the other hand, once γ exceeds 0.5, the displacement of pilot actions (signified by the mixture of colors) begin to enlarge, which is equivalent to the copilot producing an action that ignores the pilot's "intention". This captures the notion that the fidelity to the pilot decreases as we increase γ.

Algorithm 1 summarizes the procedure of applying partial forward and reverse diffusion to a pilot action a_t^s and generating a shared action a_t^s. In Section III, we investigate the fidelity-conformity trade-off by modulating γ empirically for various environments with different pilots.

III. Experiments

We evaluate our approach to shared autonomy by pairing our copilot with various pilots on a variety of simulated continuous control tasks (Fig. 4): 2D control, Lunar Lander, Lunar Reacher and Block Pushing. Each of these tasks provide the opportunity for the pilot to execute one of several different behaviors that is not known to the copilot. We design each domain to include a randomly sampled target state the pilot intends to reach (herein referred to as the *goal*). For example, in Lunar Reacher, a goal location is sampled randomly above ground. Across all tasks, we reveal the goal only to the pilot, by including it as part of the state. The copilot never has access to the goal. This results in a scenario in which pilot's intention (i.e., the goal) is unknown to the copilot.

To understand the pilot's intent, our copilot relies on a state-conditioned diffusion model. This diffusion model is trained as follows. First, we collect expert demonstrations, each containing a sequence of goal-embedded state-action pairs for each domain, where the goal is randomly sampled at each episode. We then use the resulting demonstrations to train a state-conditioned diffusion model using the DDPM loss (Eqn. 8). The details of the model architecture are described in Appendix A. As we hide a goal from our copilot, we remove goal locations from each observation prior to training.

The following experiments use a single diffusion model trained separately for each task. We note that the copilot's behavior changes only according to γ given a diffusion model.

We seek to answer the following questions: (1) How much can our copilot assist a pilot? (2) Does our copilot generalize to different pilots? and (3) What is the effect of the forward diffusion ratio γ, and how can we interpret it?

A. Continuous control domains

1) 2D Control: We build a simple 2D continuous control domain based on the maze-2D environment in D4RL [18] built on the MuJoCo simulator [48], where the goal is located either at the lower-left or lower-right corners of the large open space (Fig. 4(a)). The agent is represented as a point mass, and the actions are 2D forces applied to itself. The state consists of the agent's location and velocity, as well as the goal location. Episodes are terminated if a timeout of 300 steps is reached.

2) Lunar Lander: Lunar Lander (Fig. 4(b)) is a continuous control environment adopted from Open AI Gym [7] that involves landing a spaceship on a landing pad. The actions consist of continuous left, right and upward forces that emulate thrusters on the spaceship. The state contains the position,

TABLE I: Success and crash/out-of-bounds (OOB) rates on Lunar Lander and Lunar Reacher for different pilots with and without assistance. For each entry in the table, we evaluated 10 episodes across 30 random seeds.

Pilot	Success Rate w/o Copilot ($\gamma = 0.0$)	Success Rate w/ Copilot (ours) ($\gamma = 0.4$)	Success Rate w/ Copilot ($\gamma = 1.0$)	Crash/OOB Rate w/o Copilot ($\gamma = 0.0$)	Crash/OOB Rate w/ Copilot (ours) ($\gamma = 0.4$)	Crash/OOB Rate w/ Copilot ($\gamma = 1.0$)
Lunar Lander						
Zero	0.00 ± 0.00	26.67 ± 2.49	26.33 ± 0.47	100.00 ± 0.00	24.33 ± 2.87	18.67 ± 3.09
Random	0.00 ± 0.00	29.00 ± 2.16	20.67 ± 2.49	100.00 ± 0.00	20.00 ± 2.94	19.67 ± 4.64
Noisy	0.00 ± 0.00	$\mathbf{60.67 \pm 0.47}$	20.33 ± 2.49	63.00 ± 1.41	$\mathbf{12.00 \pm 2.16}$	18.00 ± 5.72
Laggy	6.00 ± 1.41	$\mathbf{56.33 \pm 3.30}$	23.00 ± 4.90	79.33 ± 3.09	$\mathbf{14.67 \pm 3.77}$	22.67 ± 1.25
Expert	87.33 ± 2.05	84.67 ± 3.68	22.33 ± 3.68	9.00 ± 0.82	7.00 ± 3.27	19.33 ± 0.94
Lunar Reacher						
Zero	0.33 ± 0.47	23.00 ± 3.74	33.00 ± 4.90	99.67 ± 0.47	56.00 ± 5.35	43.00 ± 4.55
Random	4.67 ± 3.77	21.33 ± 4.19	28.33 ± 4.11	95.33 ± 3.77	59.33 ± 4.78	50.33 ± 5.44
Noisy	38.00 ± 0.82	$\mathbf{58.67 \pm 3.09}$	30.00 ± 2.16	48.33 ± 3.68	$\mathbf{26.33 \pm 2.05}$	42.67 ± 2.49
Laggy	25.00 ± 6.48	$\mathbf{45.67 \pm 3.40}$	29.00 ± 3.74	73.67 ± 7.32	$\mathbf{40.67 \pm 2.87}$	47.67 ± 3.86
Expert	57.67 ± 4.19	65.00 ± 2.94	29.67 ± 3.68	36.00 ± 5.35	27.00 ± 1.63	44.00 ± 3.27

TABLE II: Correct goal and timeout rates on the Block Pushing task for different pilots with and without assistance. For each entry in the table, we evaluated 10 episodes across 30 random seeds.

Pilot	Correct Goal Rate w/o Copilot ($\gamma = 0.0$)	Correct Goal Rate w/ Copilot (ours) ($\gamma = 0.2$)	Correct Goal Rate w/ Copilot ($\gamma = 1.0$)	Timeout Rate w/o Copilot ($\gamma = 0.0$)	Timeout Rate w/ Copilot (ours) ($\gamma = 0.2$)	Timeout Rate w/ Copilot ($\gamma = 1.0$)
Zero	0.00 ± 0.00	48.00 ± 2.83	51.33 ± 6.13	100.00 ± 0.00	6.00 ± 2.45	4.00 ± 0.82
Random	0.00 ± 0.00	18.33 ± 2.36	55.33 ± 3.40	100.00 ± 0.00	65.33 ± 1.25	2.67 ± 0.47
Noisy	73.33 ± 5.31	$\mathbf{83.67 \pm 2.62}$	54.33 ± 5.44	25.33 ± 3.86	13.33 ± 3.68	3.67 ± 1.70
Laggy	59.00 ± 4.08	$\mathbf{77.67 \pm 4.99}$	52.33 ± 4.71	39.33 ± 4.11	18.33 ± 4.11	7.67 ± 3.09
Expert	99.00 ± 0.82	95.67 ± 0.94	55.67 ± 5.44	1.00 ± 0.82	4.00 ± 0.82	5.00 ± 1.63

orientation, linear and rotational velocity of the spaceship, whether each leg touches the ground, and the landing pad location (provided only to the pilot). An episode ends when the spaceship lands on the landing pad and becomes idle, crashes, flies out of bounds, or it reaches a timeout of 1000 steps.

3) Lunar Reacher: A variant of Lunar Lander adopted from previous work [43, 7], where the goal is not to land, but to reach a random target location above the ground (Fig. 4(c)). The setting otherwise matches that of Lunar Lander.

4) Block Pushing: A variant of the Simulated Pushing environment [17]. The environment (Fig. 4(d)) consists of a simulated six-DoF robot xArm6 in PyBullet [9] equipped with a small cylindrical end effector. The task is to push an object into one of two target zones in the robot's workspace. The episode terminates when the target reaches one of the two locations or the number of steps exceeds a timeout of 100. The position and orientation of the block and end effector are randomly initialized at the start of each episode, while the target locations are fixed.

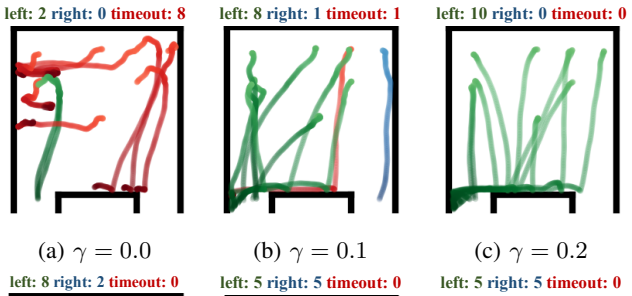
B. Pilot and copilot

1) Pilot: Our method does not require access to a pilot (surrogate or otherwise) when training the copilot. However, surrogate pilots are useful when we want to perform a large number of evaluations in a reproducible manner. Thus, we prepare two surrogate pilots consistent with previous work [39, 43]: a *Laggy* pilot and a *Noisy* pilot, both of which are corrupted versions of a single expert. At each time step, the Laggy pilot repeats its previous action with probability p_{laggy}, and otherwise executes an action drawn from the expert's policy. With probability p_{noisy}, the Noisy pilot samples an action from a uniform distribution over the action space, and otherwise executes an action sampled from the expert policy. We evaluate our shared autonomy algorithm with pilots over a broad range of p_{noisy} and p_{laggy}. Due to space constraints, we only include results for a representative subset of the policies here ($p_{\text{laggy}} = p_{\text{noisy}} = 0.8$ for 2D control, $p_{\text{laggy}} = 0.7,\ p_{\text{noisy}} = 0.5$ for Lunar Reacher, $p_{\text{laggy}} = p_{\text{noisy}} = 0.5$ for Block Pushing), but include results for the full range of pilot parameters in Figures 9–14 of Appendix B.

2) Copilot: Our copilot relies on a state-conditioned diffusion model $p_\theta(a_t \mid s_t)$ that we train for each task based on expert demonstrations. To collect demonstrations, we first train an expert policy with soft actor-critic [23] for 3M timesteps in Lunar Lander and Lunar Reacher, and 1M timesteps in Block Pushing. We then roll out the policies in each environment to collect demonstrations of state-action pairs D_{expert} for various goals. While the expert policy has access to the goal location, we remove it from the dataset of state-action pairs prior to training the state-conditioned diffusion model. We provide details of the model architecture and the hyperparameter settings in Appendix A.

C. Various pilots with our copilot

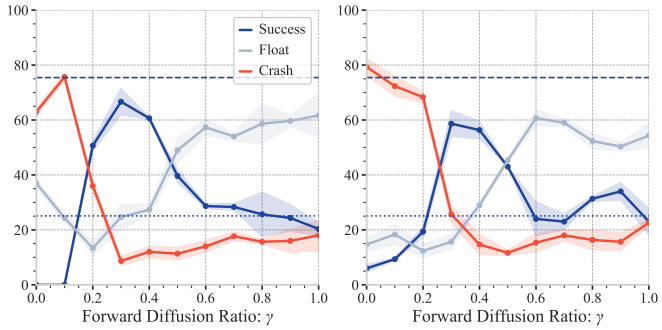
We evaluate our copilot with various surrogate pilots in all four environments. Table I shows success and crash/out-of-



(a) $\gamma = 0.0$ (b) $\gamma = 0.1$ (c) $\gamma = 0.2$

(d) $\gamma = 0.4$ (e) $\gamma = 0.6$ (f) $\gamma = 0.8$

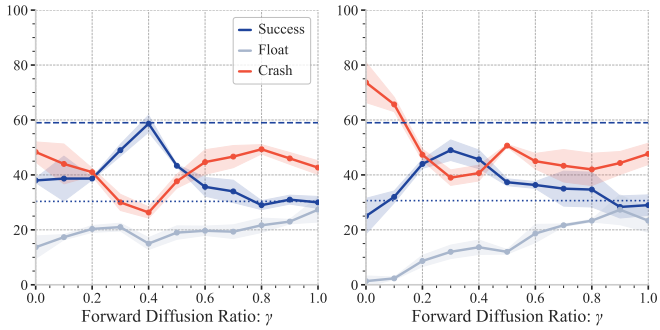
Fig. 5: A visualization of the resulting trajectories in the 2D control environment for different settings for the forward diffusion ratio γ. The user's objective is to reach the left-hand goal. Without assistance (a) the user successfully reaches the goal two times, while the eight others timeout. As we increase γ, we see that (b)–(d) the user reaches the desired goal a vast majority of the time. As γ gets closer to 1.0, (e) (f) the assisted policy conforms to the expert policy, which avoids timeouts, but without knowledge of the user's goal distributes the trajectories evenly between the left and right goals.



(a) Noisy Pilot (b) Laggy Pilot

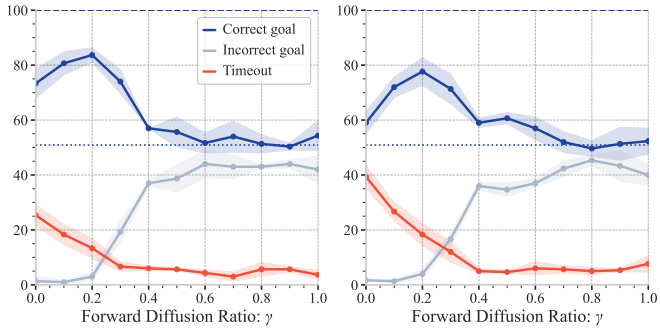
Fig. 6: Success (higher is better), floating, and crash (lower is better) rates for Lunar Lander with (a) noisy and (b) laggy pilot. The dashed blue line denotes the success rate of an expert policy, while the dotted blue line denotes the success rate of our model with full-diffusion ($\gamma = 1.0$).

bound rates when various pilots are paired with our copilot in Lunar Lander and Lunar Reacher. We see that adopting our copilot significantly improves the success and crash rates for all but the expert policy (as expected). We note that each task involves reaching or landing on a randomly sampled target that is not known to the copilot. Consequently, one can not expect the copilot to have significant effect on the success rate of the Random or Zero pilot. Nevertheless, the results show that our



(a) Noisy Pilot (b) Laggy Pilot

Fig. 7: Success, (higher is better), floating, and crash (lower is better) rates for Lunar Reacher with (a) noisy and (b) laggy pilot. The dashed blue line denotes the success rate of an expert policy, while the dotted blue line denotes the success rate of our model with full-diffusion ($\gamma = 1.0$).



(a) Noisy Pilot (b) Laggy Pilot

Fig. 8: The rates of block pushing task to terminate at correct goal, wrong goal or timeout with (a) noisy and (b) laggy pilot. The dashed blue line denotes the rate at which an expert policy reaches the correct goal, while the dotted blue line denotes the same rate for our model with full-diffusion ($\gamma = 1.0$).

copilot improves their success rates with performance similar to that of full diffusion ($\gamma = 1.0$) and significantly decreases the rate at which the pilots crash.

For the Block Pushing task, the Laggy and Noisy policy are based on an expert whose intent is always to push the block to the green zone, which we refer to as the correct goal. The set of state-action pairs on which we trained the diffusion model also include trajectories that reach the red zone. As such, without knowledge of the pilot's goal, the copilot may generate actions that push the block into the red zone (referred to as the incorrect goal), causing the episode to terminate. As we see in Table II, our copilot improves the rate at which all but an expert pilot reach the correct goal, while decreasing the rate at which episodes timeout without reaching any goal.

D. The effects of the forward diffusion ratio γ

The forward diffusion ratio γ determines how many steps of forward diffusion process to apply on pilot's action a_t^h. This value changes the balance of interpolating between source

and target distributions, which in turn controls the trade-off between fidelity and conformity of an action. In this section, we demonstrate this trade-off with different γ values in various environments. We deployed our copilot with various surrogate pilots as discussed in Section III-B for each γ value.

Figure 5 shows the effect of γ for the 2D-control domain. Although the copilot is trained on demonstrations for both goals, the pilot's goal is fixed to the left. As γ increases, we see that the number of trajectories that reach the correct goal increases. Qualitatively, we also see that the stochasticity of the trajectories lessens, and the paths towards the goal become smoother. However, when γ is too high, the copilot ignores the pilot's actions. Without knowledge of the pilot's goal, the copilot ends up distributing trajectories evenly between the left and right goals, consistent with the set of trajectories on which the model was trained.

Figures 6 and 7 summarize the results on Lunar Lander and Lunar Reacher, respectively, by categorizing the trajectories into success, crash and float. Here, float denotes a trajectory where the spaceship does not crash nor go out of bounds, but nevertheless fails to complete the task within the time limit.

In all cases, we observe that the success rate follows a distinct pattern. First, the success rate monotonically increases with γ as the copilot improves task performance. After reaching a peak success rate at a critical value of γ, the success rate gradually decreases. The drop in the success rate reflects the copilots infidelity to the user's intent. The pilot's original actions provide a signal of which goal or landing pad is correct. When γ is too large, the copilot totally ignores the pilot's actions and instead chooses to land in a manner following the copilot's original training distribution.

In general, the crash rate monotonically decreases with γ, with the exception of Lunar Reacher. This demonstrates that the quality of the generated actions improves as γ increases. In Lunar Reacher, many of the demonstrations go straight to a target, which is often placed at the edge of the the environment. Consequently, if the agent overshoots and misses the target, it is very likely that it will go out of bounds, which is counted as a crash. This is likely why increasing γ does not necessarily result in lower crash rates for Lunar Reacher.

IV. Related Work

Shared autonomy has appeared in many problem domains, including remote telepresense [11, 21, 41], assistive robotic manipulation [30, 35, 44], and assistive navigation [5, 20]. In shared autonomy, one of the most persistent challenges has been correctly identifying the pilot's intentions or goals.

Early work sidesteps this challenge by assuming a priori knowledge of the pilot's goals [10, 31]. Recent work has managed to relax this assumption by treating the pilot's goal as a latent random variable which can be inferred from environmental observations and pilot actions [1, 6, 14, 26, 28, 31, 34, 37, 53]. In spite of this forward progress, these methods still assume knowledge about some combination of the transition dynamics, the pilot's goals, or pilot's policy, making them difficult to deploy in many unstructured scenarios.

Reddy et al. [39], and subsequently [47, 43] introduce model-free deep reinforcement learning (RL) to the shared autonomy setting. Because these methods are model free, knowledge of environment dynamics is no longer required, allowing one to train a policy that is not limited to a specific model class. Several follow up works have adapted deep RL to a variety of shared autonomy problems [15, 40, 36, 8].

To the best of our knowledge, all previous work on shared autonomy with deep RL either explicitly or implicitly assumes a human-in-the-loop setting. In particular, the main training loop typically contains a step to query a pilot (i.e., human user) to obtain its action. This constraint often makes training inefficient or impractical. Chen et al. [8] addresses this issue by proposing a two-phase training scheme. In the first phrase, an autonomous agent learns a task-conditioned policy that can be helpful for assisting a pilot. This is followed by the second phase, which incorporates sparse feedback from humans. Although such approach improves the efficiency of the training pipeline, it does not change the fact that these methods inherently rely on human feedback.

Diffusion models [45] have recently been applied to many problems including image generation, image editing, text-conditioned image generation and video generation. In the field of robotics, Janner et al. [27] trained a diffusion model over trajectories, demonstrating that diffusion is capable of generating a diverse set of trajectories reaching a goal location. Anonymous [4] applies diffusion models to imitation learning. Meanwhile, Wang et al. [51] show that the policy parameterized with diffusion can generate a multi-modal distributions over possible actions. As far as we are aware, ours is the first approach that uses diffusion models in a shared control scenario, where a copilot learns from expert demonstrations and provides guidance by denoising noisy pilot action.

The core idea of our approach is in the partial forward and reverse diffusion that enables us to generate an output which effectively interpolates between the original input and a target distribution we train diffusion models on. Although we independently came up to this idea, a similar approach has been concurrently explored in image editing [33].

Finally, many recent papers have considered the problem of running reverse diffusion from some intermediate step rather than pure Gaussian noise, as we did in this paper. See for example [32, 50, 38, 52].

V. Conclusion

In this paper, we presented a new approach to shared autonomy based on diffusion models. Our approach only requires access to demonstrations that are representative of desired behavior, and does not assume access to or knowledge of the user's policy, reward feedback, or knowledge of the goal space or environment dynamics. Integral to our approach is its modulation of the forward and reverse diffusion processes in a manner that seeks to balance the user's desire to maintain control authority with the benefits of generating actions that are consistent with the distribution over desired behaviors. We evaluated our copilot on various continuous

control environments and demonstrated that our diffusion-based copilot generalizes across a variety of pilots, improving their performance, while preserving their intention. We further presented an analysis of the effects of different degrees of partial diffusion on task performance. One limitation of our approach is that there is no component that explicitly addresses the likely mismatch in state distributions between $P_{\text{pilot}}(s)$ and $P_{\text{target}}(s)$. Intuitively, it is very likely that the target state visitation distribution (i.e., expert demonstrations) is different from that of the pilot. However, our empirical results suggest that this is not a critical limitation, possibly because executing corrected actions tends to encourage the agent to visit states that are close to those visited as part of the expert demonstrations. That said, one means of addressing this is to design a goal-conditioned policy that can navigate itself to an in-distribution state at test time. We will leave this for a future work.

References

[1] Daniel Aarno, Staffan Ekvall, and Danica Kragic. Adaptive virtual fixtures for machine-assisted teleoperation tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[2] David A Abbink, Tom Carlson, Mark Mulder, Joost C.F. de Winter, Farzad Aminravan, Tricia L. Gibo, and Erwin R Boer. A topology of shared control systems—finding common ground in diversity. *IEEE Transactions on Human-Machine Systems*, 48(5), 2018.

[3] Peter Aigner and Brenan McCarragher. Human integration into robot control utilising potential fields. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1997.

[4] Anonymous. Imitating human behaviour with diffusion models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=Pv1GPQzRrC8. Under Review.

[5] Brenna D. Argall. Modular and adaptive wheelchair automation. In *Proceedings of the International Symposium on Experimental Robotics (ISER)*, 2016.

[6] Alexander Broad, Todd Murphey, and Brenna Argall. Highly parallelized data-driven MPC for minimal intervention shared control. In *Proceedings of Robotics: Science and Systems (RSS)*, 2019.

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

[8] Sean Chen, Jensen Gao, Siddharth Reddy, Glen Berseth, Anca D. Dragan, and Sergey Levine. ASHA: assistive teleoperation via human-in-the-loop reinforcement learning. *CoRR*, abs/2202.02465, 2022.

[9] Erwin Coumans and Yunfei Bai. PyBullet, a Python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

[10] Jacob W. Crandall and Michael A. Goodrich. Characterizing efficiency of human robot interaction: A case study of shared-control teleoperation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[11] Thomas Debus, Jeffrey Stoll, Robert D. Howe, and Pierre Dupont. Cooperative human and machine perception in teleoperated assembly. In *Proceedings of the International Symposium on Experimental Robotics (ISER)*, 2000.

[12] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat GANs on image synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[13] Anca D. Dragan and Siddhartha S. Srinivasa. Formalizing assistive teleoperation. In *Proceedings of Robotics: Science and Systems (RSS)*, 2012.

[14] Anca D. Dragan and Siddhartha S. Srinivasa. A policy-blending formalism for shared control. *International Journal of Robotics Research*, 32(7), 2013.

[15] Yuqing Du, Stas Tiomkin, Emre Kiciman, Daniel Polani, Pieter Abbeel, and Anca Dragan. AvE: Assistance via empowerment. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[16] Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating second-order functional knowledge for better option pricing. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, 2000.

[17] Pete Florence, Corey Lynch, Andy Zeng, Oscar Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2021.

[18] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4RL: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[19] Cristian Garcia. Denoising diffusion probabilistic models, 2021. URL https://github.com/acids-ircam/diffusion_models. Online.

[20] Mahmoud Ghorbel, Joelle Pineau, Richard Gourdeau, Shervin Javdani, and Siddhartha Srinivasa. A decision-theoretic approach for the collaborative control of a smart wheelchair. *International Journal of Robotics Research*, 10:131–145, 2017.

[21] Ray C. Goertz. Manipulators used for handling radioactive materials. *Human Factors in Technology*, 1963.

[22] Matthew C Gombolay, Reymundo A Gutierrez, Giancarlo F Sturla, and Julie A Shah. Decision-making authority, team efficiency and human worker satisfaction in mixed human—robot teams. In *Proceedings of Robotics: Science and Systems (RSS)*, 2014.

[23] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018.

[24] Kris Hauser. Recognition, prediction, and planning for assisted teleoperation of freeform tasks. *Autonomous*

Robots, 35(4), 2013.

[25] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *CoRR*, abs/2006.11239, 2020.

[26] Guy Hoffman and Cynthia Breazeal. Effects of anticipatory action on human-robot teamwork efficiency, fluency, and perception of team. In *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2007.

[27] Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2022.

[28] Shervin Javdani, Siddhartha S. Srinivasa, and J. Andrew Bagnell. Shared autonomy via hindsight optimization. *arXiv:1503.07619*, 2015.

[29] Shervin Javdani, Siddhartha S. Srinivasa, and J. Andrew Bagnell. Shared autonomy via hindsight optimization. In *Proceedings of Robotics: Science and Systems (RSS)*, 2015.

[30] Hyun K. Kim, J. Biggs, W. Schloerb, M. Carmena, Mikhail A. Lebedev, Miguel A.L. Nicolelis, and Mandayam A. Srinivasan. Continuous shared control for stabilizing reaching and grasping with brain-machine interfaces. *IEEE Transactions on Biomedical Engineering*, 53(6), 2006.

[31] Jonathan Kofman, Xianghai Wu, Timothy J Luu, and Siddharth Verma. Teleoperation of a robot manipulator using a vision-based human-robot interface. *IEEE Transactions on Industrial Electronics*, 52(5), 2005.

[32] Zhaoyang Lyu, Xu Xudong, Ceyuan Yang, Dahua Lin, and Bo Dai. Accelerating diffusion models via early stop of the diffusion process. *ArXiv*, abs/2205.12524, 2022.

[33] Chenlin Meng, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. SDEdit: Image synthesis and editing with stochastic differential equations. *CoRR*, abs/2108.01073, 2021.

[34] Katharina Muelling, Abdeslam Boularias, Betty Mohler, Bernhard Schölkopf, and Jan Peters. Learning strategies in table tennis using inverse reinforcement learning. *Biological Cybernetics*, 108(5), 2014.

[35] Katharina Muelling, Arun Venkatraman, Jean-Sebastien Valois, John E. Downey, Jeffrey Weiss, Shervin Javdani, Martial Hebert, Andrew B. Schwartz, Jennifer L. Collinger, and J. Andrew Bagnell. Autonomy infused teleoperation with application to brain computer interface controlled manipulation. *Autonomous Robots*, 41(6), August 2017.

[36] Yoojin Oh, Marc Toussaint, and Jim Mainprice. Learning to arbitrate human and robot control using disagreement between sub-policies. *CoRR*, abs/2108.10634, 2021.

[37] Claudia Pérez-D'Arpino and Julie A Shah. Fast target prediction of human reaching motion for cooperative human-robot manipulation tasks using time series classification. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[38] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. DreamFusion: Text-to-3D using 2D diffusion. *arXiv*, 2022.

[39] Siddharth Reddy, Anca Dragan, and Sergey Levine. Shared autonomy via deep reinforcement learning. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

[40] Siddharth Reddy, Sergey Levine, and Anca D. Dragan. First contact: Unsupervised human-machine co-adaptation via mutual information maximization. *ArXiv*, abs/2205.12381, 2022.

[41] Louis B. Rosenberg. Virtual fixtures: Perceptual tools for telerobotic manipulation. In *Proceedings of the Virtual Reality Annual International Symposium (VRAIS)*, 1993.

[42] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*, 2022.

[43] Charles Schaff and Matthew R. Walter. Residual policy learning for shared autonomy. In *Proceedings of Robotics: Science and Systems (RSS)*, 2020.

[44] Sebastian Schröer, Ingo Killmann, Barbara Frank, Martin Völker, Lukas Fiederer, Tonio Ball, and Wolfram Burgard. An autonomous robotic assistant for drinking. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[45] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.

[46] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vancouver, B.C., Canada, 2019.

[47] Weihao Tan, David Koleczek, Siddhant Pradhan, Nicholas Perello, Vivek Chettiar, Vishal Rohra, Aaslesha Rajaram, Soundararajan Srinivasan, H. M. Sajjad Hossain, and Yash Chandak. On optimizing interventions in shared autonomy. *CoRR*, abs/2112.09169, 2021.

[48] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012. doi: 10.1109/IROS.2012.6386109.

[49] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7): 1661–1674, 2011.

[50] Haochen Wang, Xiaodan Du, Jiahao Li, Raymond A. Yeh, and Greg Shakhnarovich. Score Jacobian chaining: Lifting pretrained 2D diffusion models for 3D generation. *arXiv preprint arXiv:2212.00774*, 2022.

[51] Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy

class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*, 2022.

[52] Wang Yinhuai, Yu Jiwen, and Zhang Jian. Zero shot image restoration using denoising diffusion null-space model. *arXiv:2212.00490*, 2022.

[53] Wentao Yu, Redwan Alqasemi, Rajiv Dubey, and Norali Pernalete. Telemanipulation assistance based on motion intention recognition. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

APPENDIX A
MODEL ARCHITECTURE AND HYPERPARAMETERS

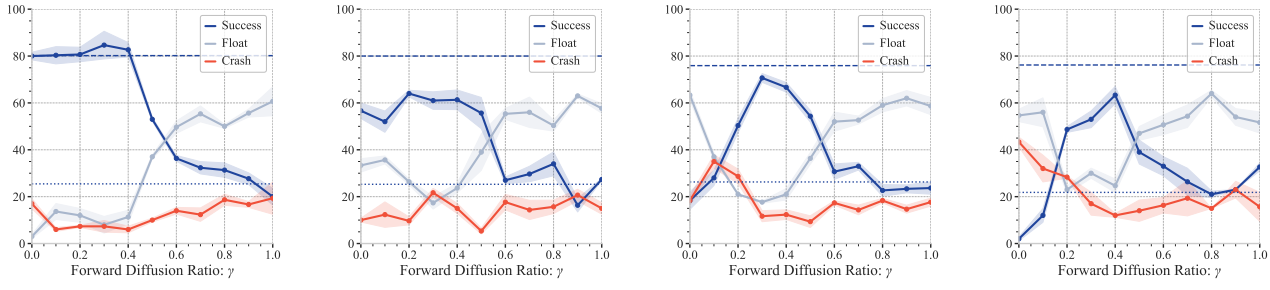
We implement the state-conditioned denoising network $\boldsymbol{\epsilon}_\theta(\boldsymbol{a}_t, k \mid \boldsymbol{s}_t)$ that makes up the diffusion model as a 4-layer multi-layer perceptron (MLP) with latent dimension h_{dim} and softplus [16] activation after every layer except for the last one. The network outputs a vector that has same dimension as in the input. Each layer is conditioned on a diffusion timestep k that looks up a corresponding embedding with dimension $\boldsymbol{h}_{\text{dim}} = 128$, and it is subsequently fused with the output of the linear layer by element-wise product. To accommodate its condition on $\boldsymbol{s}_t$, our denoising network works on a concatenation of state-action $(\boldsymbol{s}_t; \boldsymbol{a}_t)$, and during training Gaussian noise is added only to the action part to make up a target output. Namely its input $\boldsymbol{x}$ and the target $\boldsymbol{y}$ are

$$x = (\boldsymbol{s}_t; \boldsymbol{a}_t + \epsilon), \ \epsilon \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{11a}$$

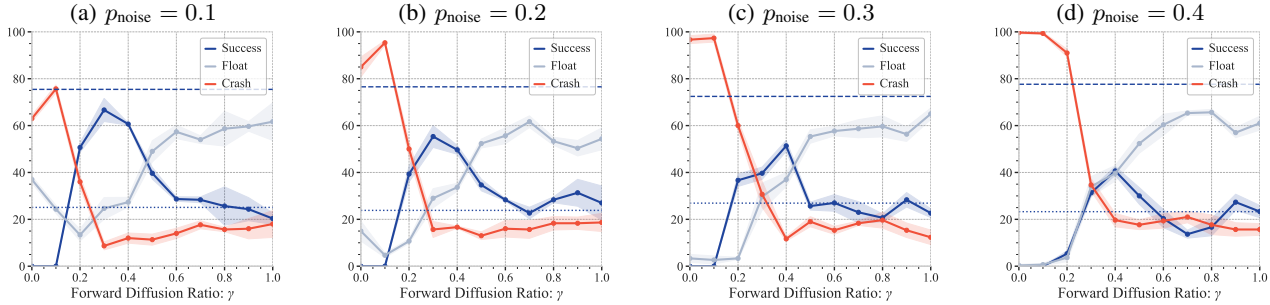
$$y = (\mathbf{0}; \boldsymbol{\epsilon}) \tag{11b}$$

We adopt a publicly available implementation of DDPM [19]. For the sake of faster action generation, we set the number of diffusion steps $K = 50$, and beta schedule $\beta_{\text{min}} = 10^{-4}, \beta_{\text{max}} = 0.26$.

APPENDIX B
EFFECT OF γ OVER DIFFERENT PROBABILITIES IN SURROGATE PILOTS

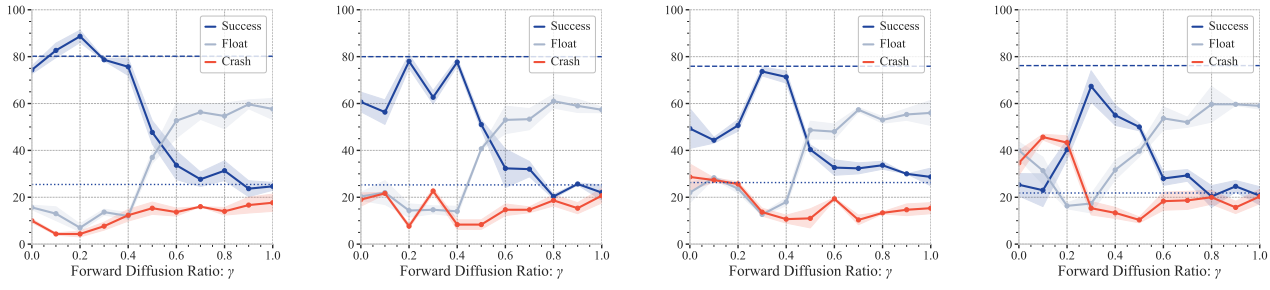


(a) $p_{\text{noise}} = 0.1$ (b) $p_{\text{noise}} = 0.2$ (c) $p_{\text{noise}} = 0.3$ (d) $p_{\text{noise}} = 0.4$

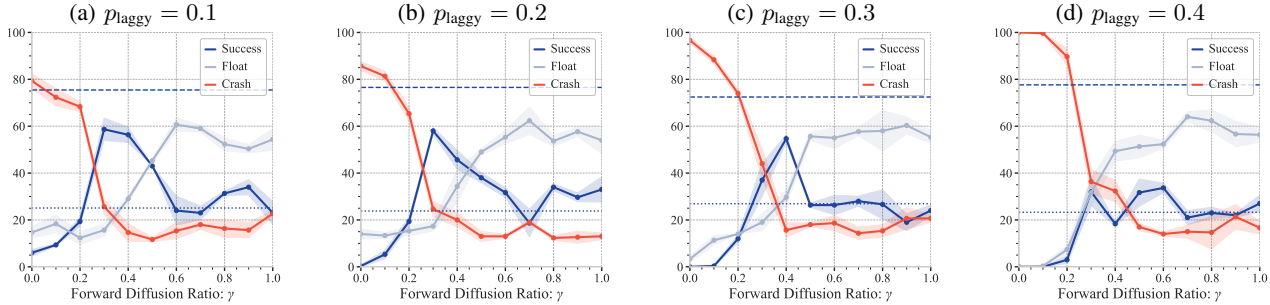


(e) $p_{\text{noise}} = 0.5$ (f) $p_{\text{noise}} = 0.6$ (g) $p_{\text{noise}} = 0.7$ (h) $p_{\text{noise}} = 0.8$

Fig. 9: Lunar Lander Performance as a function of the forward diffusion ratio γ with a noisy pilot with noise values ranging from (a) $p_{\text{noise}} = 0.1$ to (h) $p_{\text{noise}} = 0.8$. In all plots, "expert" denotes the correct rate of an expert policy.



(a) $p_{\text{laggy}} = 0.1$ (b) $p_{\text{laggy}} = 0.2$ (c) $p_{\text{laggy}} = 0.3$ (d) $p_{\text{laggy}} = 0.4$



(e) $p_{\text{laggy}} = 0.5$ (f) $p_{\text{laggy}} = 0.6$ (g) $p_{\text{laggy}} = 0.7$ (h) $p_{\text{laggy}} = 0.8$

Fig. 10: Lunar Lander Performance as a function of the forward diffusion ratio γ with a noisy pilot with noise values ranging from (a) $p_{\text{laggy}} = 0.1$ to (h) $p_{\text{laggy}} = 0.8$. In all plots, "expert" denotes the correct rate of an expert policy.

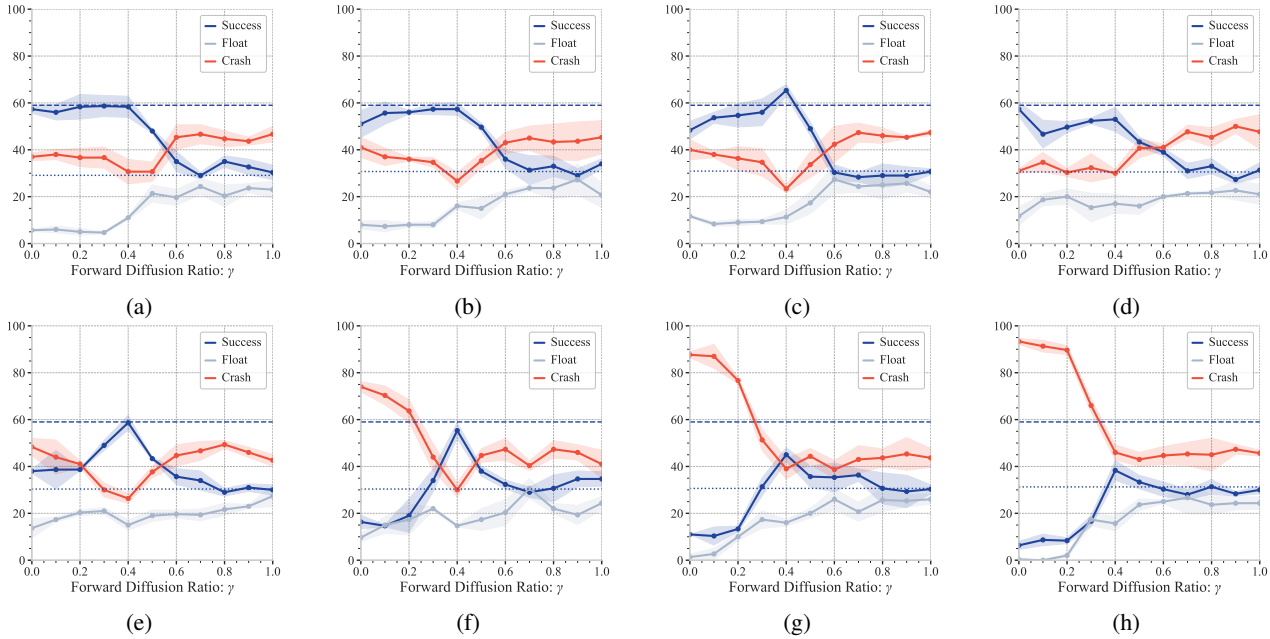


Fig. 11: Lunar Reacher Performance for Noisy Actor:The success, crash and float rates with different p_{noisy} from (a) 0.1 to (h) 0.8. In all plots, "expert" denotes the correct rate of an expert policy.

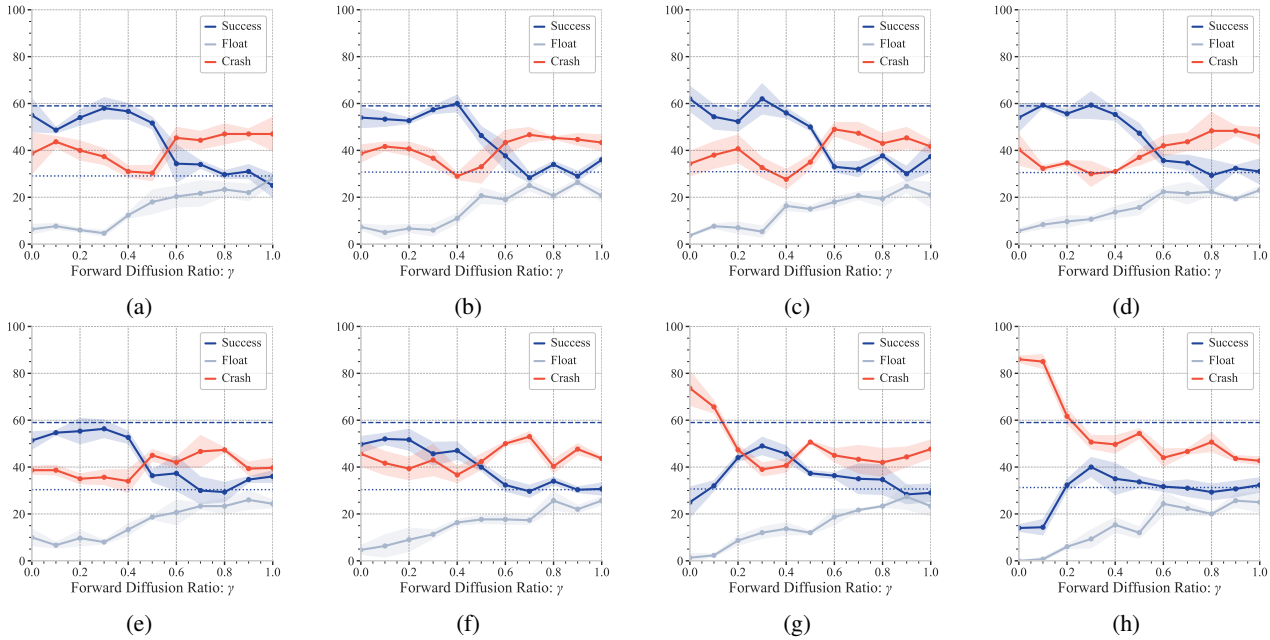


Fig. 12: Lunar Reacher Performance for Laggy Actor: The success, crash and float rates with different p_{laggy} from (a) 0.1 to (h) 0.8. In all plots, "expert" denotes the correct rate of an expert policy.

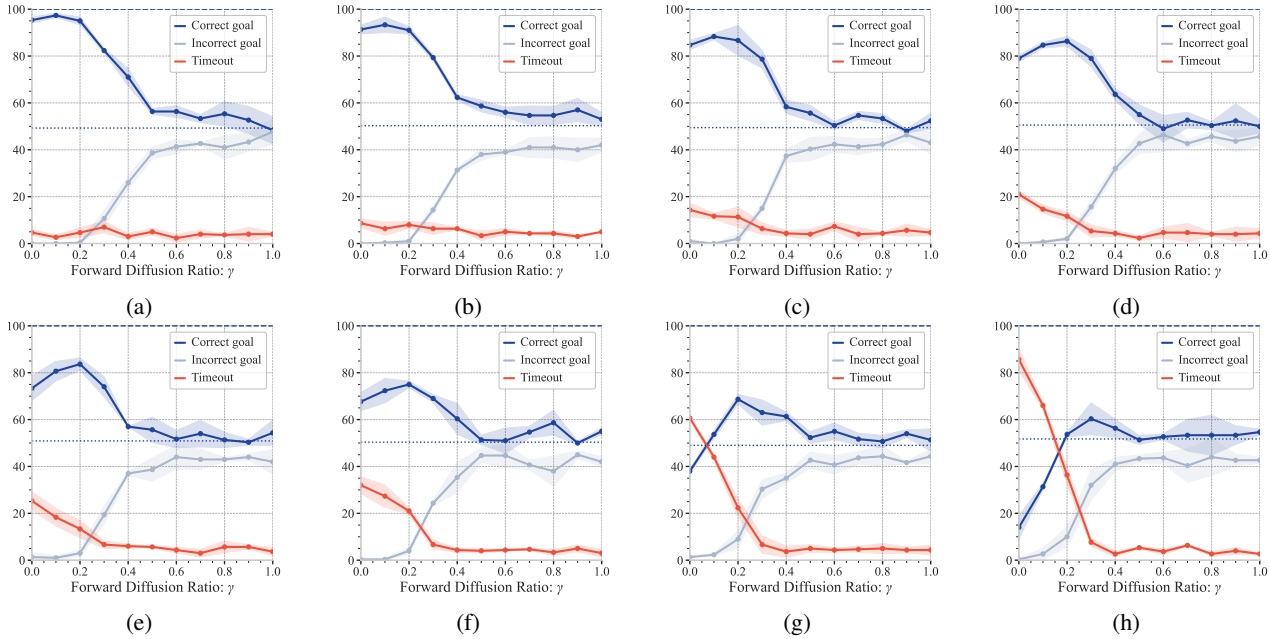


Fig. 13: Block Pushing Performance for Noisy Actor:The correct, wrong and timeout rates with different p_{noisy} from (a) 0.1 to (h) 0.8. In all plots, "expert" denotes the correct rate of an expert policy.

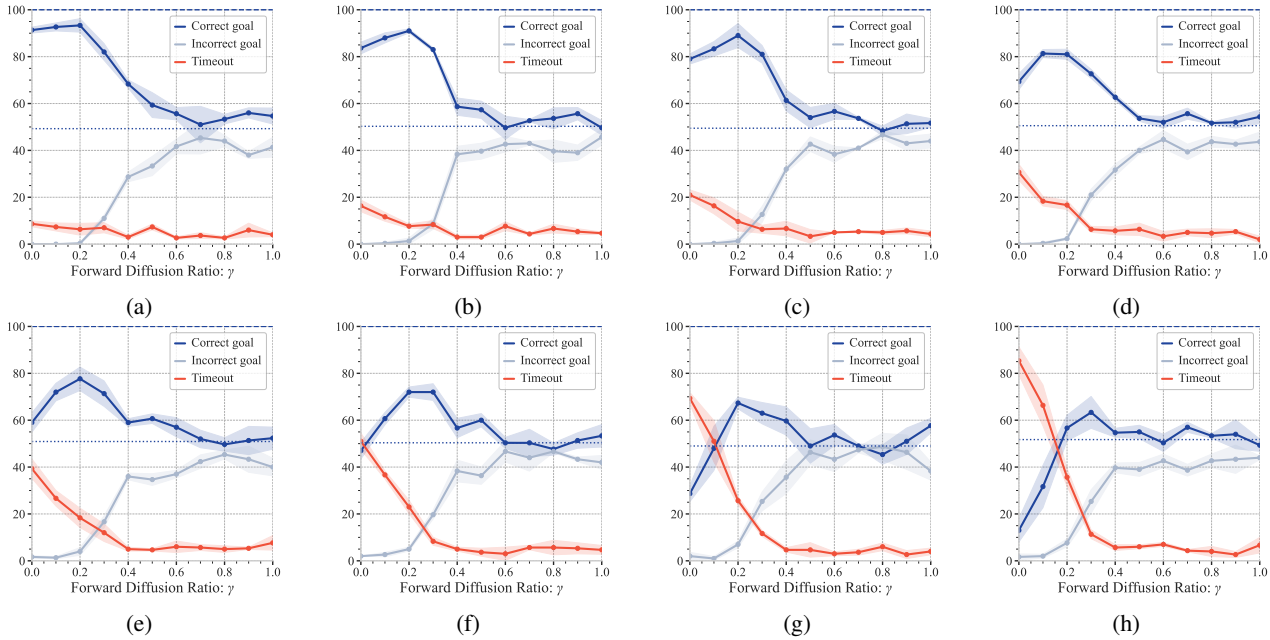


Fig. 14: Block Pushing Performance for Laggy Actor:The correct, wrong and timeout rates with different p_{laggy} from (a) 0.1 to (h) 0.8. In all plots, "expert" denotes the correct rate of an expert policy.