澳門大學
UNIVERSIDADE DE MACAU
UNIVERSITY OF MACAU

# Learning to Reach Goals via Diffusion

GongYing
gong.ying@connect.umac.mo

2024.1.5

# Contents

# 1. Introduction

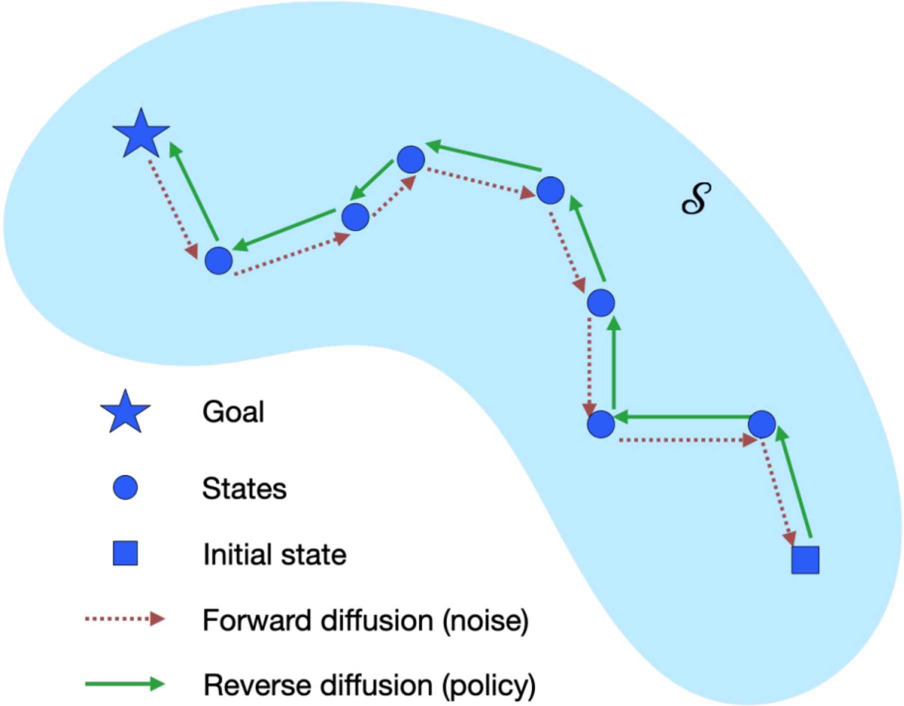
- In RL, agents learn behaviors supervised by only a reward function.
- Goal-conditioned RL (GCRL) aims to learn general policies that can reach arbitrary target states or goals within an environment requiring no extensive retraining. But its disadvantage is sparse rewards.
- Thus combining GCRL and offline RL is good for generalization and data efficiency.

**Challenges:**

- However, in offline RL many methods rely on value function, which estimates the expected discounted return associated with a given state-action pair. During training, policies generate actions not in the offline dataset (for without interaction), leading to inaccuracies and even diverging policies.
- To solve the problem of limited state-goal pairs, hindsight relabeling is employed. But it only generate state-goal pairs within the same trajectory, resulting in over-fitting.

# 1. Introduction

To solve the problems mentioned, the paper draws inspiration from diffusion models.
They construct trajectories that move away (noise) from desired goals during the learning process. Then a policy is trained to reverse (denoise) the trajectories. Thus the policy learns to reach any predefined goal state to from arbitrary initial states.

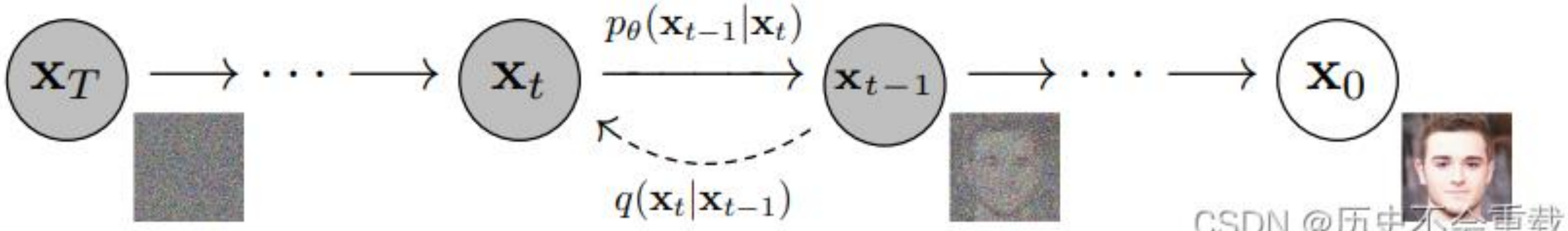


# 2. Preliminaries

## I. Diffusion probabilistic models

- The forward diffusion process is to add Gaussian noise to the real image $X_0$ at each timestep, leading to the final noised image $X_T$. At timestep t, a constant $\beta_t$ is given, we have:

$$q(X_t|X_{t-1}) = N(X_t; \sqrt{1-\beta_t}X_{t-1}, \beta_t I),$$
$$q(X_{1:T}|X_0) = \prod_{t=1}^{T} q(X_t|X_{t-1}).$$



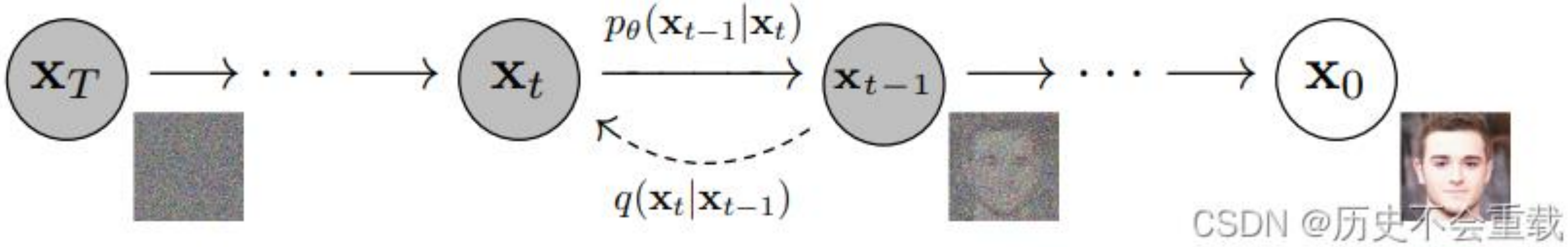
# 2. Preliminaries

## I. Diffusion probabilistic models

- Then a denoising function is learned to reverse the forward diffusion process.

$$p_\theta(X_{t-1}|X_t) = N(X_{t-1};\ \mu_\theta(X_t,t), \Sigma_\theta(X_t,t)),$$
$$p_\theta(X_{0:T}|X_0) = p(X_T)\prod_{t=1}^{T} p_\theta(X_{t-1}|X_t),$$

where $\mu_\theta$ and $\Sigma_\theta$ can be neural networks.



# 2. Preliminaries

## II. Goal-conditioned RL

- The RL problem can be described using a Markov Decision Process (MDP), denoted by $(S, A, P, r, \mu, \gamma)$, where $\mu(s)$ is the initial state distribution.
- While Goal-conditioned RL additionally considers a goal space $G = \{\phi(s) | s \in S\}$ where $\phi: S \rightarrow G$ is a known state-to-goal mapping. Now reward function depends on the goal and can be sparse and binary defined as $r(s, a, g) = \mathbb{1}[||\phi(s) - g||_2^2 \leq \delta]$, where $\delta$ is some threshold distance.
- A goal-conditioned policy is denoted by $\pi: S \times G \rightarrow A$, and given a distribution over desired goals $p(g)$, an optimal policy $\pi^\star$ aims to maximize the expected return:

$$J(\pi) = \mathbb{E}_{g \sim p(g), s_0 \sim \mu(s_0), a_t \sim \pi(\cdot | s_t, g), s_{t+1} \sim P(\cdot | s_t, a_t)}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, g)].$$

# 3. Reaching Goals via Diffusion

Consider a goal-augmented MDP $(S, A, G, P, r, \mu, \gamma)$ with goals $g \in G$ sampled from unknown goal distribution $g \sim p(g)$. Goal-conditioned RL aims to learn a policy that can learn an optimal path from any state $s \in S$ to the desired goal $g$.

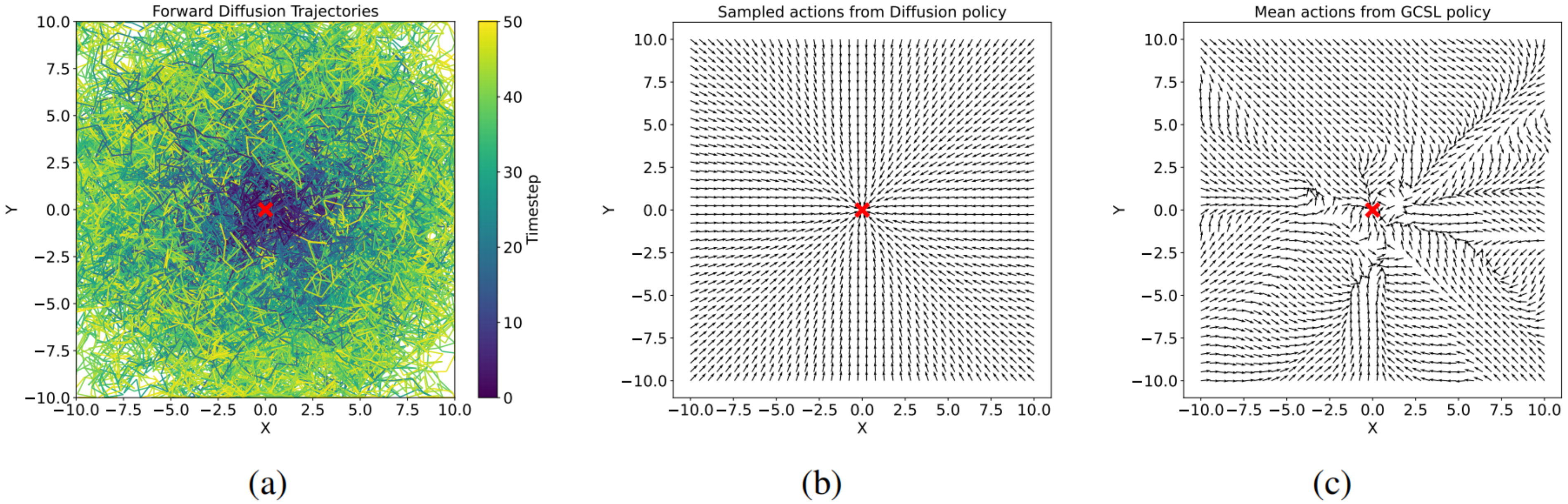



Figure 2: (a) Visualization of trajectories starting from the goal **X** generated during the forward process, (b) Predicted actions from policy trained via diffusion, (c) Predicted actions from policy trained using GCSL.

# 3. Reaching Goals via Diffusion

During training, the time horizon indicates the time difference between the current and desired goal states.
For $h$ = 1, the policy always takes the most direct path to the goal regardless of the input state. For larger values of the time horizon, the policy has a high variance close to the goal and a low variance for the optimal action further away.

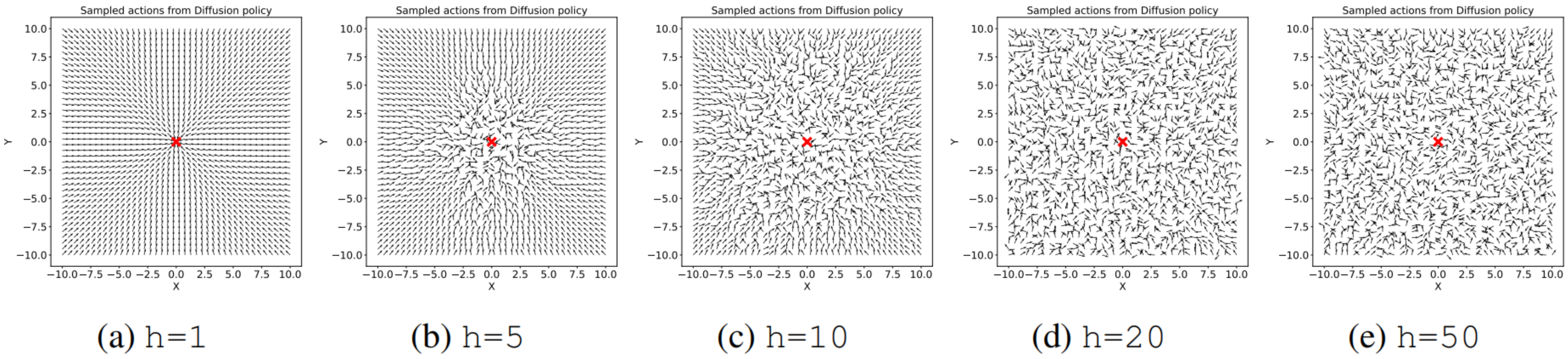



Figure 3: Actions sampled from the trained policy, showcasing the effect of time horizon during evaluation.

# 4. Goal-conditioned Diffusion Policy

**Nearest-neighbor trajectory stitching**

The forward diffusion constructs trajectories walking away from the goal to provide training data for the policy. In order for this strategy to be effective, we want to generate as many state-goal pairs as possible to help the policy generalize well. Hindsight relabeling can generate positive goal-conditioned observations by replacing the desired goals with achieved goals (e.g. A robot is supposed to move a ball from A to B, but it moved it to C. Then we can assume the task is from A to C, leading to a successful task).

**Algorithm 1** Nearest-neighbor Trajectory Stitching

**Input:** Dataset $\mathcal{D}$, distance threshold $\delta$, number of new trajectories to collect $M$
**Output:** Augmented dataset $\mathcal{D}_{\text{new}}$
$\mathcal{D}_{\text{new}} \leftarrow \mathcal{D}$
Construct ball tree $T$ for all states
**for** $m \leftarrow 1, \dots, M$ **do**
  Sample random final state $s_T$ from $\mathcal{D}$
  $\tau_{\text{new}} \leftarrow \{s_T\}$
  $s_{\text{current}} \leftarrow s_T$
  **for** $t \leftarrow T, \dots, 1$ **do**
    $s_{\text{neighbor}}, d \leftarrow T.\text{query}(s_{\text{current}}, k = 1)$
    **if** $d \leq \delta$ **then**
      Add preceding $(s_{\text{prev}}, a_{\text{prev}})$ from $s_{\text{neighbor}}$ to $\tau_{\text{new}}$
    **else**
      Add preceding $(s_{\text{prev}}, a_{\text{prev}})$ from $s_{\text{current}}$ to $\tau_{\text{new}}$
    **end if**
    $s_{\text{current}} \leftarrow s_{\text{prev}}$
  **end for**
  $\mathcal{D}_{\text{new}} \leftarrow \mathcal{D}_{\text{new}} \cup \tau_{\text{new}}$
**end for**
**Return:** $\mathcal{D}_{\text{new}}$

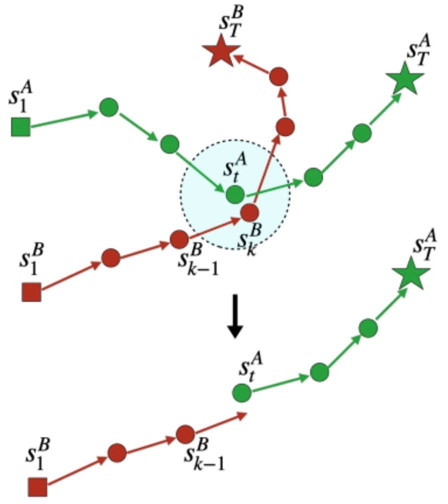


Figure 5: Trajectory stitching.

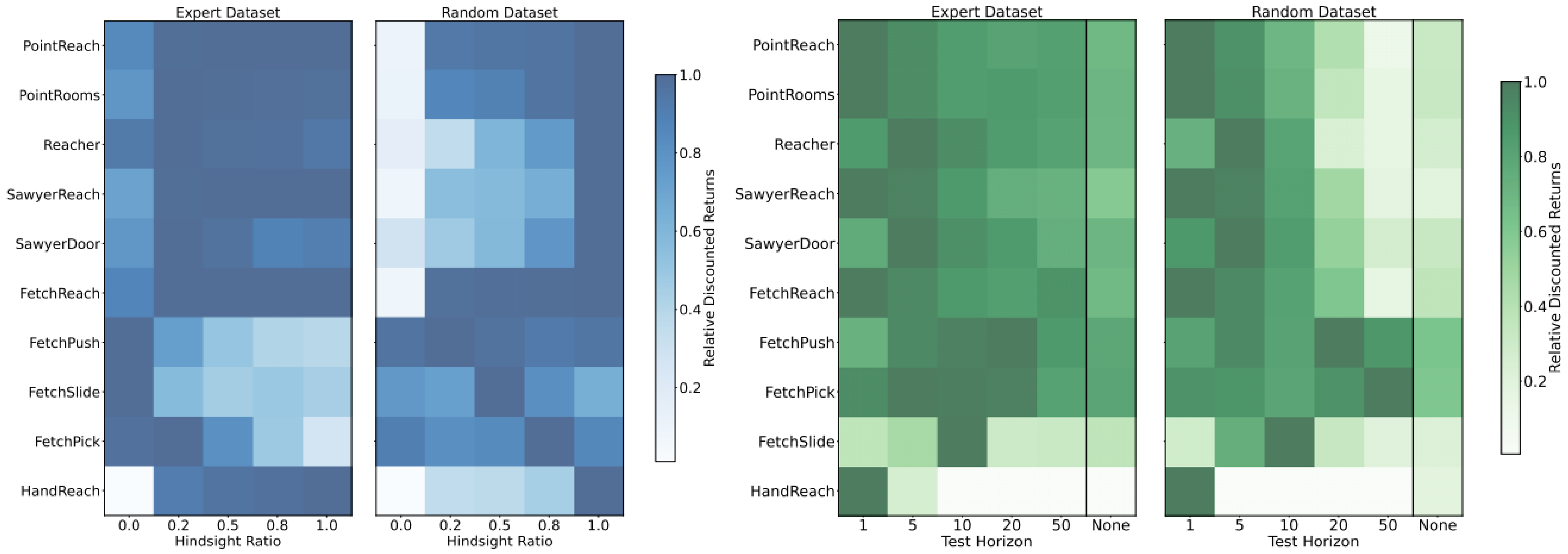
# 5. Experiments

Table 1: Discounted returns, averaged over 10 seeds.

| | Task Name | Ours | | | Offline GCRL | | | | Diffusion-based | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Merlin | Merlin-P | Merlin-NP | GoFAR | WGCSL | GCSL | AM | DD | g-DQL |
| **Expert** | PointReach | **29.26**±0.04 | 29.17±0.15 | **29.30**±0.05 | 27.18±0.65 | 25.91±0.87 | 22.85±1.26 | 26.14±1.11 | 10.03±0.88 | 28.65±0.44 |
| | PointRooms | 25.38±0.37 | 25.25±0.07 | **25.42**±0.32 | 20.40±1.00 | 19.90±0.99 | 18.28±2.29 | 23.24±1.58 | 5.84±2.67 | **27.53**±0.57 |
| | Reacher | 22.75±0.59 | 23.25±0.17 | **24.97**±0.54 | 22.51±0.82 | **23.35**±0.64 | 20.05±1.37 | 22.36±1.03 | 4.39±1.08 | 22.54±1.42 |
| | SawyerReach | **26.89**±0.07 | 25.05±0.60 | **27.35**±0.06 | 22.82±1.15 | 22.07±1.46 | 19.20±1.79 | 23.56±0.33 | 3.39±0.75 | 24.17±0.01 |
| | SawyerDoor | **26.18**±2.19 | 25.75±0.97 | 26.15±2.08 | 23.62±0.35 | 23.92±1.10 | 20.12±1.33 | **26.39**±0.42 | 7.85±0.77 | 24.81±0.38 |
| | FetchReach | **30.29**±0.03 | 30.26±0.02 | **30.42**±0.04 | 29.21±0.26 | 28.17±0.38 | 23.68±1.07 | 29.08±0.12 | 1.55±0.68 | 28.71±0.15 |
| | FetchPush | 19.91±1.20 | 2.23±2.20 | 21.58±1.63 | **22.41**±1.69 | **22.22**±1.51 | 17.58±1.47 | 19.86±3.16 | 5.49±2.85 | 17.82±0.55 |
| | FetchPick | 19.66±0.78 | 1.43±1.01 | **20.41**±0.92 | **19.79**±1.12 | 18.32±1.56 | 12.95±1.90 | 17.04±3.81 | 2.76±0.64 | 14.45±0.61 |
| | FetchSlide | 4.19±1.89 | 0.00±0.00 | **4.95**±2.02 | 3.34±1.01 | **5.17**±3.17 | 1.67±1.41 | 3.31±1.46 | 1.21±0.59 | 0.98±0.59 |
| | HandReach | **22.11**±0.55 | 0.00±0.00 | **24.93**±0.49 | 15.39±6.37 | 18.05±5.12 | 0.15±0.11 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 |
| | Average Rank | **2.7** | 5.3 | **1.6** | 4.5 | 4.6 | 7.3 | 5.0 | 8.3 | 5.1 |
| **Random** | PointReach | **29.26**±0.04 | 29.21±0.08 | **29.31**±0.04 | 23.96±0.93 | 25.76±0.96 | 17.74±1.84 | 25.55±0.57 | 10.12±0.72 | 22.65±1.57 |
| | PointRooms | **24.80**±0.36 | 24.07±0.19 | **25.16**±0.59 | 18.09±4.13 | 19.41±1.01 | 14.69±2.51 | 19.10±1.39 | 5.76±2.99 | 20.88±0.96 |
| | Reacher | 21.09±0.65 | 16.65±0.48 | 22.24±0.54 | **25.10**±0.68 | 22.98±0.91 | 10.62±2.30 | **23.70**±0.62 | 4.74±0.36 | 6.06±0.84 |
| | SawyerReach | **26.70**±0.14 | 25.46±0.12 | **26.86**±0.07 | 19.48±1.39 | 21.32±1.40 | 8.78±2.59 | 25.29±0.35 | 3.46±0.86 | 2.84±0.05 |
| | SawyerDoor | 19.05±0.66 | 18.26±1.18 | **21.69**±2.36 | **20.69**±2.14 | 19.58±3.55 | 12.47±3.08 | 10.82±1.67 | 7.92±0.86 | 14.77±0.51 |
| | FetchReach | **30.42**±0.04 | 30.38±0.02 | **30.42**±0.04 | 28.34±0.98 | 27.94±0.30 | 18.96±1.77 | 27.11±0.22 | 1.71±0.77 | 1.21±0.46 |
| | FetchPush | 5.21±0.43 | 5.08±0.32 | **7.22**±0.35 | **6.99**±1.27 | 5.35±3.36 | 4.22±2.19 | 4.53±1.94 | 4.49±1.34 | 5.35±0.23 |
| | FetchPick | 3.75±0.18 | 3.02±0.16 | **4.36**±0.19 | **3.81**±3.71 | 1.87±1.59 | 0.81±0.82 | 3.08±1.35 | 2.16±0.75 | 2.17±0.18 |
| | FetchSlide | **2.67**±0.35 | 0.00±0.00 | **3.15**±0.14 | 1.32±1.22 | 1.04±0.98 | 0.24±0.27 | 1.12±0.39 | 1.31±0.52 | 0.00±0.00 |
| | HandReach | **14.89**±2.54 | 0.00±0.00 | **17.61**±3.06 | 0.08±0.07 | 2.54±1.42 | 1.41±0.51 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 |
| | Average Rank | **2.8** | 4.8 | **1.3** | 3.8 | 4.5 | 7.3 | 5.3 | 7.7 | 6.7 |

Maximum trajectory length=50, reward=1 if goal is reached (else 0).
Merlin-P uses a proposed learned parametric reverse dynamics model and reverse policy, to generate additional diffusion trajectories. Merlin-NP uses the trajectory stitching method.

# 5. Experiments




(a) (b)

Figure 6: Discounted returns for each dataset with different values of (a) hindsight ratio and (b) time horizon during evaluation. Values are normalized with respect to the maximum value in each row.

# Thank you.