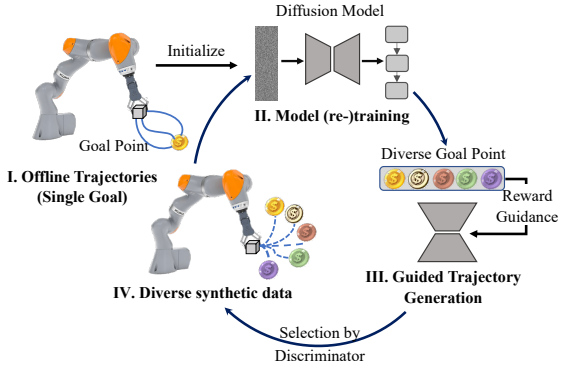
AdaptDiffuser: Diffusion Models as Adaptive Self-evolving Planners

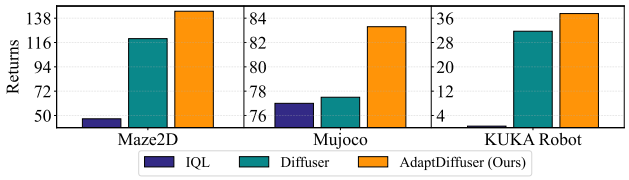
Zhixuan Liang [1] **Yao Mu** [1] **Mingyu Ding** [1 2] **Fei Ni** [3] **Masayoshi Tomizuka** [2] **Ping Luo** [1 4]

Abstract

Diffusion models have demonstrated their powerful generative capability in many tasks, with great potential to serve as a paradigm for offline reinforcement learning. However, the quality of the diffusion model is limited by the insufficient diversity of training data, which hinders the performance of planning and the generalizability to new tasks. This paper introduces AdaptDiffuser, an evolutionary planning method with diffusion that can self-evolve to improve the diffusion model hence a better planner, not only for seen tasks but can also adapt to unseen tasks. AdaptDiffuser enables the generation of rich synthetic expert data for goal-conditioned tasks using guidance from reward gradients. It then selects high-quality data via a discriminator to finetune the diffusion model, which improves the generalization ability to unseen tasks. Empirical experiments on two benchmark environments and two carefully designed unseen tasks in KUKA industrial robot arm and Maze2D environments demonstrate the effectiveness of AdaptDiffuser. For example, AdaptDiffuser not only outperforms the previous art Diffuser (Janner et al., 2022) by 20.8% on Maze2D and 7.5% on MuJoCo locomotion, but also adapts better to new tasks, e.g., KUKA pick-and-place, by 27.9% without requiring additional expert data. More visualization results and demo videos could be found on our project page.



(a) Illustration of AdaptDiffuser.



(b) Performance comparisons on three benchmarks.

Figure 1. Overall framework and performance comparison of AdaptDiffuser. It enables diffusion models to generate rich synthetic expert data using guidance from reward gradients of either seen or unseen goal-conditioned tasks. Then, it iteratively selects high-quality data via a discriminator to finetune the diffusion model for self-evolving, leading to improved performance on seen tasks and better generalizability to unseen tasks.

1. Introduction

Offline reinforcement learning (RL) (Levine et al., 2020; Prudencio et al., 2022) aims to learn policies from previously collected offline data without interacting with the live environment. Traditional offline RL approaches require fitting value functions or computing policy gradients, which are challenging due to limited offline data (Agarwal et al., 2020; Kumar et al., 2020; Wu et al., 2019; Kidambi et al., 2020). Recent advances in generative sequence modeling (Chen et al., 2021a; Janner et al., 2021; 2022) provide effective alternatives to conventional RL problems by modeling the joint distribution of sequences of states, actions, rewards and values. For example, Decision Transformer (Chen et al., 2021a) casts offline RL as a form of conditional sequence modeling, which allows more efficient and stable learning without the need to train policies via traditional RL algorithms like temporal difference learning (Sutton, 1988). By treating RL as a sequence modeling problem, it bypasses the need of bootstrapping for long-term credit assignment,

[1]Department of Computer Science, The University of Hong Kong, Hong Kong SAR [2]University of California, Berkeley, USA [3]College of Intelligence and Computing, Tianjin University, Tianjin, China [4]Shanghai AI Laboratory, Shanghai, China. Correspondence to: Ping Luo <pluo.lhi@gmail.com>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

avoiding one of the "deadly triad" (Sutton & Barto, 2018) challenges in reinforcement learning.

Therefore, devising an excellent sequence modeling algorithm is essential for the new generation of offline RL. The diffusion probability model (Rombach et al., 2022; Ramesh et al., 2022), with its demonstrated success in generative sequence modeling for natural language processing and computer vision, presents an ideal fit for this endeavor. It also shows great potential as a paradigm for planning and decision-making. For example, diffusion-based planning methods (Janner et al., 2022; Ajay et al., 2023; Wang et al., 2023) train trajectory diffusion models based on offline data and apply flexible constraints on generated trajectories through reward guidance during sampling. In consequence, diffusion planners show notable performance superiority compared with transformer-based planners like Decision Transformer (Chen et al., 2021a) and Trajectory Transformer (Janner et al., 2021) on long horizon tasks, while enabling goal-conditioned rather than reward-maximizing control at the same time.

While diffusion-based planners have achieved success in certain areas, their performance is limited by the lack of diversity in their training data. In decision-making tasks, the cost of collecting a diverse set of offline training data may be high, and this insufficient diversity would impede the ability of the diffusion model to accurately capture the dynamics of the environment and the behavior policy. As a result, diffusion models tend to perform inferior when expert data is insufficient, and particularly when facing new tasks. This raises a natural question: can we use the generated heterogeneous data by the reward-guided diffusion model to improve the diffusion model itself since it has powerful generative sequence modeling capability? As diffusion-based planners can generate quite diverse "dream" trajectories for multiple tasks which may be different from the original task the training data are sampled from, greatly superior to Decision Transformer (Chen et al., 2021a), enabling the diffusion model to be self-evolutionary makes it a stronger planner, potentially benefiting more decision-making requirements and downstream tasks.

In this paper, we present AdaptDiffuser, a diffusion-based planner for goal-conditioned tasks that can generalize to novel settings and scenarios through self-evolution (see Figure 1). Unlike conventional approaches that rely heavily on specific expert data, AdaptDiffuser uses gradient of reinforcement learning rewards, directly integrated into the sampling process, as guidance to generate diverse and heterogeneous synthetic demonstration data for both existing and unseen tasks. The generated demonstration data is then filtered by a discriminator, of which the high-quality ones are used to fine-tune the diffusion model, resulting in a better planner with significantly improved self-bootstrapping capabilities on previously seen tasks and an enhanced ability of generalizing to new tasks. As a consequence, AdaptDiffuser not only improves the performance of the diffusion-based planner on existing benchmarks, but also enables it to adapt to unseen tasks without the need for additional expert data.

It's non-trivial to construct and evaluate AdaptDiffuser for both seen and unseen tasks.We first conduct empirical experiments on two widely-used benchmarks (MuJoCo (Todorov et al., 2012) and Maze2d) of D4RL (Fu et al., 2020) to verify the self-bootstrapping capability of AdaptDiffuser on seen tasks. Additionally, we creatively design new pick-and-place tasks based on previous stacking tasks in the KUKA (Schreiber et al., 2010) industrial robot arm environment, and introduce novel auxiliary tasks (e.g., collecting gold coins) in Maze2D. The newly proposed tasks and settings provide an effective evaluation of the generalization capabilities of AdaptDiffuser on unseen tasks.

Our contributions are three-fold: **1)** We present AdaptDiffuser, allowing diffusion-based planners to self-evolve for offline RL by generating high-quality heterogeneous data with reward-integrated diffusion model directly and filtering out inappropriate examples with a discriminator. **2)** We apply our self-evolutionary AdaptDiffuser to unseen (zero-shot) tasks without any additional expert data, demonstrating its strong generalization ability and adaptability. **3)** Extensive experiments on two widely-used offline RL benchmarks from D4RL as well as our carefully designed unseen tasks in KUKA and Maze2d environments validate the effectiveness of AdaptDiffuser.

2. Related Works

Offline Reinforcement Learning. Offline RL (Levine et al., 2020; Prudencio et al., 2022) is a popular research field that aims to learn behaviors using only offline data such as those collected from previous experiments or human demonstrations, without the need to interact with the live environment from time to time at the training stage.

However, in practice, offline RL faces a major challenge that standard off-policy RL methods may fail due to the overestimation of values, caused by the distribution deviation between the offline dataset and the policy to learn. Most conventional offline RL methods use action-space constraints or value pessimism (Buckman et al., 2021) to overcome the challenge (Agarwal et al., 2020; Kumar et al., 2020; Siegel et al., 2020; Wu et al., 2019; Yang et al., 2022). For example, conservative Q-learning (CQL) (Kumar et al., 2020) addresses these limitations by learning a conservative Q-function, ensuring the expected value under this Q-function is lower than its true value.

Reinforcement Learning as Sequence Modeling. Recently, a new paradigm for Reinforcement Learning (RL)

has emerged, in which RL is viewed as a generic sequence modeling problem. It utilizes transformer-style models to model trajectories of states, actions, rewards and values, and turns its prediction capability into a policy that leads to high rewards. As a representative, Decision Transformer (DT) (Chen et al., 2021a) leverages a causally masked transformer to predict the optimal action, which is conditional on an autoregressive model that takes the past state, action, and expected return (reward) into account. It allows the model to consider the long-term consequences of its actions when making a decision. And based on DT, Trajectory Transformer (TT) (Janner et al., 2021) is proposed to utilize transformer architecture to model distributions over trajectories, repurposes beam search as a planning algorithm, and shows great flexibility across long-horizon dynamics prediction, imitation learning, goal-conditioned RL, and offline RL. Bootstrapped Transformer (Wang et al., 2022) further incorporates the idea of bootstrapping into DT and uses the learned model to self-generate more offline data to further improve sequence model training. However, Bootstrapped Transformer could not integrate RL reward into the data synthesizing process directly and can only amplify homogeneous data trivially for its original task, which can boost the performance but cannot enhance the adaptability on another unseen task. Besides, such approaches lack flexibility in adapting to new reward functions and tasks in different environments, as the generated data is not suitable for use in new tasks or environments.

Diffuser (Janner et al., 2022) presents a powerful framework for trajectory generation using the diffusion probabilistic model, which allows the application of flexible constraints on generated trajectories through reward guidance during sampling. The consequent work, Decision Diffuser (Ajay et al., 2023) introduces conditional diffusion with reward or constraint guidance for decision-making tasks, further enhancing Diffuser's performance. Additionally, Diffusion-QL (Wang et al., 2023), adds a regularization term to the training loss of the conditional diffusion model, guiding the model to learn optimal actions. Nevertheless, the performance of these methods is still limited by the quality of offline expert data, leaving room for improvement in adapting to new tasks or settings.

Diffusion Probabilistic Model. Diffusion models are a type of generative model that represents the process of generating data as an iterative denoising procedure (Sohl-Dickstein et al., 2015; Ho et al., 2020). They have made breakthroughs in multiple tasks such as image generation (Song et al., 2021), waveform generation (Chen et al., 2021b), 3D shape generation (Zhou et al., 2021) and text generation (Austin et al., 2021). These models, which learn the latent structure of the dataset by modeling the way in which data points diffuse through the latent space, are closely related to score matching (Hyvärinen, 2005) and energy-based models (EBMs) (LeCun et al., 2006; Du & Mordatch, 2019; Nijkamp et al., 2019; Grathwohl et al., 2020), as the denoising process can be seen as a form of parameterizing the gradients of the data distribution (Song & Ermon, 2019).

Moreover, in the sampling process, diffusion models allow flexible conditioning (Dhariwal & Nichol, 2021) and have the ability to generate compositional behaviors (Du et al., 2020). It shows that diffusion models own promising potential to generate effective behaviors from diverse datasets and plan under different reward functions including those not encountered during training.

3. Preliminary

Reinforcement Learning is generally modeled as a Markov Decision Process (MDP) with a fully observable state space, denoted as $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space. Besides, $\mathcal{T}$ is the state transition function with the dynamics of this discrete-time system that $\boldsymbol{s}_{t+1} = \mathcal{T}(\boldsymbol{s}_t, \boldsymbol{a}_t)$ at state $\boldsymbol{s}_t \in \mathcal{S}$ given the action $\boldsymbol{a}_t \in \mathcal{A}$. $\mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t)$ defines the reward function and $\gamma \in (0, 1]$ is the discount factor for future reward.

Considering the offline reinforcement learning as a sequence modeling task, the objective of trajectory optimization is to find the optimal sequence of actions $\boldsymbol{a}^*_{0:T}$ that maximizes the expected return with planning horizon T, which is the sum of per time-step rewards or costs $R(\boldsymbol{s}_t, \boldsymbol{a}_t)$:

$$\boldsymbol{a}^*_{0:T} = \arg\max_{\boldsymbol{a}_{0:T}} \mathcal{J}(\boldsymbol{s}_0, \boldsymbol{a}_{0:T}) = \arg\max_{\boldsymbol{a}_{0:T}} \sum_{t=0}^{T} \gamma^t R(\boldsymbol{s}_t, \boldsymbol{a}_t). \tag{1}$$

The sequence data generation methods utilizing diffusion probabilistic models (Sohl-Dickstein et al., 2015; Ho et al., 2020) pose the generation process as an iterative denoising procedure, denoted by $p_\theta(\boldsymbol{\tau}^{i-1} \mid \boldsymbol{\tau}^i)$ where $\boldsymbol{\tau}$ represents a sequence and i is an indicator of the diffusion timestep.

Then the distribution of sequence data is expanded with the step-wise conditional probabilities of the denoising process,

$$p_\theta\left(\boldsymbol{\tau}^0\right) = \int p\left(\boldsymbol{\tau}^N\right) \prod_{i=1}^{N} p_\theta\left(\boldsymbol{\tau}^{i-1} \mid \boldsymbol{\tau}^i\right) \mathrm{d}\boldsymbol{\tau}^{1:N} \tag{2}$$

where $p\left(\boldsymbol{\tau}^N\right)$ is a standard normal distribution and $\boldsymbol{\tau}^0$ denotes original (noiseless) sequence data.

The parameters θ of the diffusion model are optimized by minimizing the evidence lower bound (ELBO) of negative log-likelihood of $p_\theta\left(\boldsymbol{\tau}^0\right)$, similar to the techniques used in variational Bayesian methods.

$$\theta^* = \arg\min_\theta -\mathbb{E}_{\boldsymbol{\tau}^0}\left[\log p_\theta\left(\boldsymbol{\tau}^0\right)\right] \tag{3}$$

What's more, as the denoising process is the reverse of a forward diffusion process which corrupts input data by grad-

ually adding noise and is typically denoted by $q\left(\boldsymbol{\tau}^i \mid \boldsymbol{\tau}^{i-1}\right)$, the reverse process can be parameterized as Gaussian under the condition that the forward process obeys the normal distribution and the variance is small enough (Feller, 2015).

$$p_\theta\left(\boldsymbol{\tau}^{i-1} \mid \boldsymbol{\tau}^i\right) = \mathcal{N}\left(\boldsymbol{\tau}^{i-1} \mid \mu_\theta\left(\boldsymbol{\tau}^i, i\right), \Sigma^i\right) \tag{4}$$

in which μ_θ and Σ are the mean and covariance of the Gaussian distribution respectively.

For model training, with the basis on Eq. 3 and 4, (Ho et al., 2020) proposes a simplified surrogate loss:

$$\mathcal{L}_{\text{denoise}}(\theta) := \mathbb{E}_{i,\boldsymbol{\tau}^0\sim q,\epsilon\sim\mathcal{N}}[||\epsilon - \epsilon_\theta(\boldsymbol{\tau}^i, i)||^2] \tag{5}$$

where $i \in \{0, 1, ..., N\}$ is the diffusion timestep, $\epsilon \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ is the target noise, and $\boldsymbol{\tau}^i$ is the trajectory $\boldsymbol{\tau}^0$ corrupted by noise ϵ for i times. This is equivalent to predicting the mean μ_θ of $p_\theta\left(\boldsymbol{\tau}^{i-1} \mid \boldsymbol{\tau}^i\right)$ as the function mapping from $\epsilon_\theta(\boldsymbol{\tau}^i, i)$ to $\mu_\theta(\boldsymbol{\tau}^i, i)$ is a closed-form expression.

4. Method

In this section, we first introduce the basic planning with the diffusion method and its limitations. Then, we propose AdaptDiffuser, a novel self-evolved sequence modeling method for decision-making with the basis of diffusion probabilistic models. AdaptDiffuser is designed to enhance the performance of diffusion models in existing decision-making tasks, especially the goal-conditioned tasks, and further improve their adaptability in unseen tasks without any expert data to supervise the training process.

4.1. Planning with Task-oriented Diffusion Model

Following previous work (Janner et al., 2022), we can redefine the planning trajectory as a special kind of sequence data with actions as an additional dimension of states like:

$$\boldsymbol{\tau} = \begin{bmatrix} \boldsymbol{s}_0 & \boldsymbol{s}_1 & ... & \boldsymbol{s}_T \\ \boldsymbol{a}_0 & \boldsymbol{a}_1 & ... & \boldsymbol{a}_T \end{bmatrix} \tag{6}$$

Then we can use the diffusion probabilistic model to perform trajectory generation. However, the aim of planning is not to restore the original trajectory but to predict future actions with the highest reward-to-go, the offline reinforcement learning should be formulated as a conditional generative problem with guided diffusion models that have achieved great success on image synthesis (Dhariwal & Nichol, 2021). So, we drive the conditional diffusion process:

$$q(\boldsymbol{\tau}^{i+1}|\boldsymbol{\tau}^i), \quad p_\theta(\boldsymbol{\tau}^{i-1}|\boldsymbol{\tau}^i, \boldsymbol{y}(\boldsymbol{\tau})) \tag{7}$$

where the new term $\boldsymbol{y}(\boldsymbol{\tau})$ is some specific information of the given trajectory $\boldsymbol{\tau}$, such as the reward-to-go (return) $\mathcal{J}(\boldsymbol{\tau}^0)$ of the trajectory, the constraints that must be satisfied by

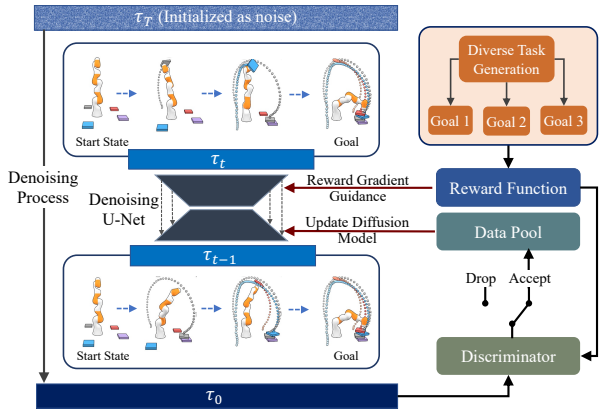


Figure 2. Overall framework of AdaptDiffuser. To improve the adaptability of the diffusion model to diverse tasks, rich data with distinct objectives is generated, guided by each task's reward function. During the diffusion denoising process, we utilize a pre-trained denoising U-Net to progressively generate high-quality trajectories. At each denoising time step, we take the task-specific reward of a trajectory to adjust the gradient of state and action sequence, thereby creating trajectories that align with specific task objectives. Subsequently, the generated synthetic trajectory is evaluated by a discriminator to see if it meets the standards. If yes, it is incorporated into a data pool to fine-tune the diffusion model. The procedure iteratively enhances the generalizability of our model for both seen and unseen settings.

the trajectory and so on. On this basis, we can rewrite the optimization objective as,

$$\theta^* = \arg\min_\theta -\mathbb{E}_{\boldsymbol{\tau}^0}\left[\log p_\theta(\boldsymbol{\tau}^0|\boldsymbol{y}(\boldsymbol{\tau}^0))\right] \tag{8}$$

Therefore, for tasks aiming to maximize the reward-to-go, we take $\mathcal{O}_t$ to denote the optimality of the trajectory at timestep t. And $\mathcal{O}_t$ obeys Bernoulli distribution with $p(\mathcal{O}_t = 1) = \exp(\gamma^t \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t))$. When $p(\mathcal{O}_{1:T} \mid \boldsymbol{\tau}^i)$ meets specific Lipschitz conditions, the conditional transition probability of the reverse diffusion process can be approximated as (Feller, 2015):

$$p_\theta(\boldsymbol{\tau}^{i-1} \mid \boldsymbol{\tau}^i, \mathcal{O}_{1:T}) \approx \mathcal{N}(\boldsymbol{\tau}^{i-1}; \mu_\theta + \alpha\Sigma g, \Sigma) \tag{9}$$

where, $g = \nabla_{\boldsymbol{\tau}} \log p(\mathcal{O}_{1:T} \mid \boldsymbol{\tau})|_{\boldsymbol{\tau}=\mu_\theta}$

$$= \sum_{t=0}^{T} \gamma^t \nabla_{\boldsymbol{s}_t,\boldsymbol{a}_t} \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t)|_{(\boldsymbol{s}_t,\boldsymbol{a}_t)=\mu_t} = \nabla_{\boldsymbol{\tau}} \mathcal{J}(\mu_\theta).$$

Besides, for tasks aiming to satisfy single point conditional constraint (e.g. goal conditioned tasks), the constraint can be simplified by substituting conditional values for the sampled values of all diffusion timesteps $i \in \{0, 1, ..., N\}$.

Although this paradigm has achieved competitive results with previous planning methods which are not based on diffusion models, it only performs conditional guidance during

the reverse diffusion process and assumes the unconditional diffusion model is trained perfectly over the forward process. However, as depicted in Eq. 9, the quality of generated trajectory $\boldsymbol{\tau}$ depends not only on the guided gradient g but more on the learned means μ_θ and covariance Σ of the unconditional diffusion model. If the learned μ_θ deviates far from the optimal trajectory, no matter how strong the guidance g is, the final generated result will be highly biased and of low quality. Then, learning from Eq. 5, the quality of μ_θ hinges on the training data, the quality of which, however, is uneven across different tasks, especially on unseen tasks. Previous diffusion-based planning methods have not solved the problem which limits the performance of these methods on both existing and unseen tasks, and thus have poor adaptation ability.

4.2. Self-evolved Planning with Diffusion

Therefore, with the aim to improve the adaptability of these planners, we propose AdaptDiffuser, a novel self-evolved decision-making approach based on diffusion probabilistic models, to enhance the quality of the trained means μ_θ and covariance Σ of the forward diffusion process. AdaptDiffuser relies on self-evolved synthetic data generation to enrich the training dataset which is denoted as $\boldsymbol{\tau}_0$ and synthetic data fine-tuning to boost performance. After that, AdaptDiffuser follows the paradigm depicted in Eq. 9 to find the optimal action sequence for the given task with the guidance of reward gradients.

As shown in Figure 2, to implement AdaptDiffuser, we firstly generate a large number of synthetic demonstration data for unseen tasks which do not exist in the training dataset in order to simulate a wide range of scenarios and behaviors that the diffusion model may encounter in the real world. This synthetic data is iteratively generated through the sampling process of the original diffusion probabilistic model θ_0^* with reward guidance, taking the advantage of its great generation ability. We will discuss the details of the synthetic data generation in Section 4.3 and here we just abbreviate it as a function $\mathcal{G}(\mu_\theta, \Sigma, \nabla_{\boldsymbol{\tau}}\mathcal{J}(\mu_\theta))$.

Secondly, we design a rule-based discriminator $\mathcal{D}$, with reward and dynamics consistency guidance, to select high-quality data from the generated data pool. Previous sequence modeling methods which predict the rewards $\mathcal{R}(\boldsymbol{s}, \boldsymbol{a})$ simultaneously with generated states and actions are unable to solve the dynamics consistency problem that the actual next state with transition model $\boldsymbol{s}' = \mathcal{T}(\boldsymbol{s}, \boldsymbol{a})$ greatly deviates from the predicted next state. What's more, these deviated trajectories are taken as feasible solutions under previous settings.

To resolve this problem, AdaptDiffuser only takes the state sequence $\boldsymbol{s} = [\boldsymbol{s}_0, \boldsymbol{s}_1, ..., \boldsymbol{s}_T]$ of the generated trajectory and then performs state tracking control using a traditional or neural network-based inverse dynamics model $\mathcal{I}$ to derive real executable actions, denoted as $\widetilde{\boldsymbol{a}}_t = \mathcal{I}(\boldsymbol{s}_t, \boldsymbol{s}_{t+1})$. This step ensures the action that does not violate the robot's dynamic constraints. After that, AdaptDiffuser performs $\widetilde{\boldsymbol{a}}_t$ to obtain the revised next state $\widetilde{\boldsymbol{s}}_{t+1} = \mathcal{T}(\widetilde{\boldsymbol{s}}_t, \widetilde{\boldsymbol{a}}_t)$, and then filters out the trajectories whose revised state $\widetilde{\boldsymbol{s}}_{t+1}$ has a too large difference from the generated $\boldsymbol{s}_{t+1}$ (measured by MSE $d = ||\widetilde{\boldsymbol{s}}_{t+1} - \boldsymbol{s}_{t+1}||_2$). The remaining trajectories $\widetilde{\boldsymbol{s}}$ are then used to predict the reward by $\widetilde{\mathcal{R}} = \mathcal{R}(\widetilde{\boldsymbol{s}}, \widetilde{\boldsymbol{a}})$ with the new actions $\widetilde{\boldsymbol{a}}$ and are selected according to this reward. In this way, we can derive high-quality synthetic data to fine-tune the diffusion probabilistic model.

We repeat this process multiple times in order to continually improve the model's performance and adapt it to new tasks, ultimately improving its generalization performance. So, it can be formulated as,

$$\begin{aligned} \theta_k^* &= \arg\min_\theta -\mathbb{E}_{\hat{\boldsymbol{\tau}}_k}\left[\log p_\theta(\hat{\boldsymbol{\tau}}_k | \boldsymbol{y}(\hat{\boldsymbol{\tau}}_k))\right] \\ \boldsymbol{\tau}_{k+1} &= \mathcal{G}\left(\mu_{\theta_k^*}, \Sigma, \nabla_{\boldsymbol{\tau}}\mathcal{J}(\mu_{\theta_k^*})\right) \\ \hat{\boldsymbol{\tau}}_{k+1} &= [\hat{\boldsymbol{\tau}_k}, \mathcal{D}(\widetilde{\mathcal{R}}(\boldsymbol{\tau}_{k+1}))] \end{aligned} \tag{10}$$

where $k \in \{0, 1, ...\}$ is the number of iteration rounds and the initial dataset $\hat{\boldsymbol{\tau}}_0 = \boldsymbol{\tau}_0$.

4.3. Reward-guided Synthetic Data Generation

To improve the performance and adaptability of the diffusion probabilistic model on unseen tasks, we need to generate synthetic trajectory data using the learned diffusion model at the current iteration. We achieve it by defining a series of tasks with different goals and reward functions.

Continuous Reward Function. For the tasks with continuous reward function, represented by MuJoCo (Todorov et al., 2012), we follow the settings that define a binary random variable indicating the optimality with probability mapped from a continuous value, to convert the reward maximization problem to a continuous optimization problem. We can easily take Eq. 9 to generate synthetic results.

Sparse Reward Function. The reward function of tasks as typified by a goal-conditioned problem like Maze2D is a unit step function $\mathcal{J}(\boldsymbol{\tau}) = \boldsymbol{\chi}_{\boldsymbol{s}_g}(\boldsymbol{\tau})$ whose value is equal to 1 if and only if the generated trajectory contains the goal state $\boldsymbol{s}_g$. The gradient of this reward function is Dirac delta function (Zhang, 2021) which is not a classical function and cannot be adopted as guidance. However, if it is considered from the perspective of taking the limit, the constraint can be simplified as replacing all corresponding sampled values with constraints over the diffusion timesteps.

Combination. Many realistic tasks need these two sorts of reward functions simultaneously. For example, if there exists an auxiliary task in Maze2D environment that requires the planner to not only find a way from the start point to

the goal point but also collect the gold coin in the maze. This task is more difficult and it's infeasible to add this constraint to the sparse reward term because there is no idea about which timestep the generated trajectory should pass the additional reward point (denoted as $\boldsymbol{s}_c$). As a solution, we propose to combine these two sorts of methods and define an auxiliary reward guiding function to satisfy the constraints.

$$\mathcal{J}(\boldsymbol{\tau}) = \sum_{t=0}^{T} ||\boldsymbol{s}_t - \boldsymbol{s}_c||_p \tag{11}$$

where p represents p-norm. Then, with Eq. 11 we plug it into Eq. 9 as the marginal probability density function and force the last state of the generated trajectory $\boldsymbol{\tau}^0$ to be $\boldsymbol{s}_c$. The generated trajectories that meet the desired criteria of the discriminator are added to the set of training data for the diffusion model learning as synthetic expert data. This process is repeated multiple times until a sufficient amount of synthetic data has been generated. By iteratively generating and selecting high-quality data based on the guidance of expected return and dynamics transition constraints, we can boost the performance and enhance the adaptability of the diffusion probabilistic model.

5. Experiment

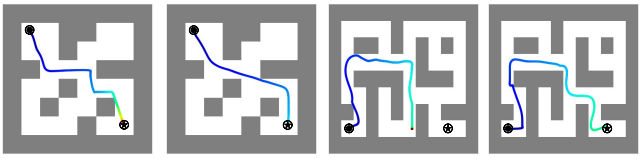
5.1. Benchmarks

Maze2D: Maze2D (Fu et al., 2020) environment is a navigation task in which a 2D agent needs to traverse from a randomly designated location to a fixed goal location where a reward of 1 is given. No reward shaping is provided at any other location. The objective of this task is to evaluate the ability of offline RL algorithms to combine previously collected sub-trajectories in order to find the shortest path to the evaluation goal. Three maze layouts are available: "umaze", "medium", and "large". The expert data for this task is generated by selecting random goal locations and using a planner to generate sequences of waypoints that are followed by using a PD controller to perform dynamic tracking. We also provide a method to derive more diverse layouts with ChatGPT in Appendix G.

MuJoCo: MuJoCo (Todorov et al., 2012) is a physics engine that allows for real-time simulation of complex mechanical systems. It has three typical tasks: Hopper, HalfCheetah, and Walker2d. Each task has 4 types of datasets to test the performance of an algorithm: "medium", "random", "medium-replay" and "medium-expert". The "medium" dataset is created by training a policy with a certain algorithm and collecting 1M samples. The "random" dataset is created by using a randomly initialized policy. The "medium-replay" dataset includes all samples recorded during training until the policy reaches a certain level of performance. There is also a "medium-expert" dataset which is a mix of expert demonstrations and sub-optimal data.

Table 1. **Offline Reinforcement Learning Performance in Maze2d Environment.** We show the results of AdaptDiffuser and previous planning methods to validate the bootstrapping effect of our method on a goal-conditioned task.

Environment	MPPI	CQL	IQL	Diffuser	AdaptDiffuser
U-Maze	33.2	5.7	47.4	113.9	**135.1** ± 5.8
Medium	10.2	5.0	34.9	121.5	**129.9** ± 4.6
Large	5.1	12.5	58.6	123.0	**167.9** ± 5.0
Average	16.2	7.7	47.0	119.5	**144.3**



(a) Diffuser (b) AdaptDiffuser (c) Diffuser (*Failed*) (d) AdaptDiffuser

Hard Case 1 (Maze2D-Medium) Hard Case 2 (Maze2D-Large)

Figure 3. **Hard Cases of Maze2D with Long Planning Path.** Paths are generated in the Maze2D environment with a specified start ◉ and goal ✪ condition.

KUKA Robot: The KUKA Robot (Schreiber et al., 2010) benchmark is a standardized evaluation tool that is self-designed to measure the capabilities of a robot arm equipped with a suction cup at the end. It consists of two tasks: conditional stacking (Janner et al., 2022) and pick-and-place. More details can be seen in Sec. 5.3.2. By successfully completing these tasks, the KUKA Robot benchmark can accurately assess the performance of the robot arm and assist developers in improving its design.

5.2. Performance Enhancement on Existing Tasks

5.2.1. Experiments on Maze2D Environment

Overall Performance. Navigation in Maze2D environment takes planners hundreds of steps to reach the goal location. Even the best model-free algorithms have to make great efforts to adequately perform credit assignments and reliably reach the target. We plan with AdaptDiffuser using the strategy of sparse reward function to condition on the start and goal location. We compare our method with the best model-free algorithms (IQL Kostrikov et al. 2022 and CQL Kumar et al. 2020), conventional trajectory optimizer MPPI (Williams et al., 2015) and previous diffusion-based approach Diffuser (Janner et al., 2022) in Table 1. This comparison is fair because model-free methods can also identify the location of the goal point which is the only state with a non-zero reward.

As shown in Table 1, scores achieved by AdaptDiffuser are over 125 in all maze sizes and are 20 points higher than those of Diffuser in average, indicating our method's strong effectiveness in goal-conditioned tasks.

Table 2. **Offline Reinforcement Learning Performance in MuJoCo Environment.** We report normalized average returns of D4RL tasks (Fu et al., 2020) in the table. And the mean and the standard error are calculated over 3 random seeds.

Dataset	Environment	BC	CQL	IQL	DT	TT	MOPO	MOReL	MBOP	Diffuser	AdaptDiffuser
Med-Expert	HalfCheetah	55.2	91.6	86.7	86.8	95.0	63.3	53.3	**105.9**	88.9	89.6 ±0.8
Med-Expert	Hopper	52.5	105.4	91.5	107.6	**110.0**	23.7	108.7	55.1	103.3	**111.6** ±2.0
Med-Expert	Walker2d	**107.5**	**108.8**	**109.6**	**108.1**	101.9	44.6	95.6	70.2	**106.9**	**108.2** ±0.8
Medium	HalfCheetah	42.6	44.0	**47.4**	42.6	**46.9**	42.3	42.1	44.6	42.8	44.2 ±0.6
Medium	Hopper	52.9	58.5	66.3	67.6	61.1	28.0	**95.4**	48.8	74.3	**96.6** ±2.7
Medium	Walker2d	75.3	72.5	78.3	74.0	79.0	17.8	77.8	41.0	79.6	**84.4** ±2.6
Med-Replay	HalfCheetah	36.6	**45.5**	**44.2**	36.6	41.9	53.1	40.2	42.3	37.7	38.3 ±0.9
Med-Replay	Hopper	18.1	95.0	94.7	82.7	91.5	67.5	**93.6**	12.4	**93.6**	**92.2** ±1.5
Med-Replay	Walker2d	26.0	77.2	73.9	66.6	82.6	39.0	49.8	9.7	70.6	**84.7** ±3.1
Average		51.9	77.6	77.0	74.7	78.9	42.1	72.9	47.8	77.5	**83.4**

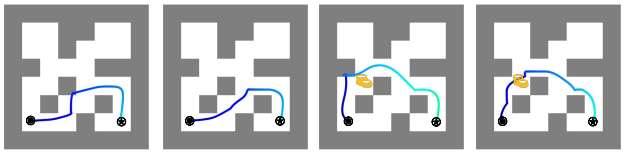
Visualization of Hard Cases. In order to more intuitively reflect the improvement of our method compared with previous Diffuser (Janner et al., 2022), we select one difficult planning example of Maze2D-Medium and one of Maze2D-Large respectively for visualization, as shown in Figure 3. Among the Maze2D planning paths with sparse rewards, the example with the longest path to be planned is the hardest one. Therefore, in Maze2D-Medium (Fig. 3 (a) (b)), we designate the start point as (1, 1) with goal point (6, 6), while in Maze2D-Large (Fig. 3 (c) (d)), we specify the start point as (1, 7) with goal point (9, 7) in the figure.

It can be observed from Fig. 3 that in Hard Case 1, AdaptDiffuser generates a shorter and smoother path than that generated by Diffuser. So, AdaptDiffuser achieves a larger reward. And in Hard Case 2, previous Diffuser method even fails to plan while our AdaptDiffuser derives a feasible path.

5.2.2. Experiments on MuJoCo Environment

MuJoCo tasks are employed to test the performance enhancement of our AdaptDiffuser learned from heterogeneous data of varying quality using the publicly available D4RL datasets (Fu et al., 2020). We evaluate our approach with a number of existing algorithms that cover a variety of data-driven methodologies, including model-free RL algorithms like CQL (Kumar et al., 2020) and IQL (Kostrikov et al., 2022); return-conditioning approaches like Decision Transformer (DT) (Chen et al., 2021a); and model-based RL algorithms like Trajectory Transformer (TT) (Janner et al., 2021), MOPO (Yu et al., 2020), MOReL (Kidambi et al., 2020), and MBOP (Argenson & Dulac-Arnold, 2021). The results are shown in Table 2. Besides, it is also worth noting that in the MuJoCo environment, the state sequence $\widetilde{\boldsymbol{s}}$ derived by taking the generated actions $\boldsymbol{a}$ is very close to the generated state sequence $\boldsymbol{s}$, so we directly use $\widetilde{\mathcal{R}}(\boldsymbol{s}, \boldsymbol{a}) = \mathcal{R}(\boldsymbol{s}, \boldsymbol{a})$ in this dataset.

Observed from the table, our method AdaptDiffuser is either competitive or outperforms most of the offline RL baselines across all three different locomotion settings. And more importantly, compared with Diffuser (Janner et al., 2022), our method achieves higher reward in almost all the datasets and improves the performance greatly, especially in "Hopper-Medium" and "Walker2d-Medium" environments. We analyze that this is because the quality of the original data in the "Medium dataset" is poor, so AdaptDiffuser has an evident effect on improving the quality of the training dataset, thus significantly enhancing the performance of the planner based on the diffusion probabilistic model. The results of the "Medium-Expert" dataset verify this analysis because the quality of original data in the "Medium-Expert" dataset (especially the Halfcheetah environment) has been good enough, making the generation of new data only has a little gain on the model performance.



(a) Diffuser (*Collision*) (b) AdaptDiffuser (c) Diffuser (*Collision*) (d) AdaptDiffuser (*No Collision*)

Original Setting With Gold Coin on (4, 2)

Figure 4. **Maze2d Navigation with Gold Coin Picking Task.** Subfigures (a) (b) show the optimal path when there are no gold coins in the Maze. (The generated routes walk at the bottom of the Maze.) And subfigures (c) (d) add additional reward in (4,2) position of the Maze. The planners generate new paths that pass through the gold coin as shown in subfigures (c) (d). (The newly generated routes walk in the middle of the maze.)

5.3. Adaptation Ability on Unseen Tasks

5.3.1. Maze2D with Gold Coin Picking Task

On top of existing Maze2D settings, we carefully design a new task that requires the agent to navigate as well as pick all gold coins in the maze. We show an example with an additional reward in (4, 2) in Figure 4.

Table 3. **Adaptation Performance on Pick-and-Place Task**

Environment	Diffuser	AdaptDiffuser
Pick and Place setup 1	28.16 ± 2.0	**36.03** ± 2.1
Pick and Place setup 2	35.25 ± 1.4	**39.00** ± 1.3
Average	31.71	**37.52**

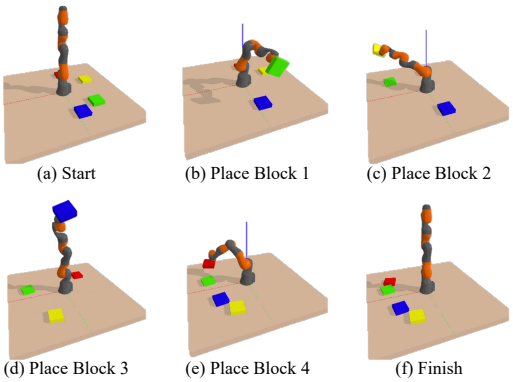


Figure 5. **Visualization of KUKA Pick-and-Place Task.** We require the KUKA Arm to move the blocks from their random initialized positions on the right side of the table to the left and arrange them in the order of yellow, blue, green, and red (from near to far).

We can see that when there is no additional reward, both Diffuser (Janner et al., 2022) and our method AdaptDiffuser choose the shorter path at the bottom of the figure to reach the goal point. But, when additional reward is added in the (4, 2) position of the maze, both planners change to the path walking in the middle of the figure under the guidance of rewards. However, at this time, the path generated by Diffuser causes the agent to collide with the wall, while AdaptDiffuser generates a smoother collision-free path, reflecting the superiority of our method.

5.3.2. KUKA Pick and Place Task

Task Specification. There are two tasks in the KUKA robot arm environment. One is the conditional stacking task, as defined in (Janner et al., 2022), where the robot must correctly stack blocks in a predetermined order on a designated location, using blocks that have been randomly placed. And the other is the pick-and-place task designed by us, which aims to place the randomly initialized blocks in their own target locations in a predetermined order. The reward functions of both tasks are defined as one upon successful placements and zero otherwise.

To test the adaptation capability of AdaptDiffuser and other baselines, we only provide expert trajectory data for the conditional stacking task, which is generated by PDDLStream (Garrett et al., 2020), but we require the planner to generalize to pick-and-put task without any expert data. The performance of the pick-and-place task is supposed to be a good measure of the planner's adaptability.

Table 4. **Ablation on Iterative Phases.** The mean and the standard error are calculated over 3 random seeds.

Dataset	Environment	1st Phase	2nd Phase
Medium-Expert	HalfCheetah	**89.3** ± 0.6	**89.6** ± 0.8
Medium-Expert	Hopper	**110.7** ± 3.2	**111.6** ± 2.0
Medium-Expert	Walker2d	**107.7** ± 0.9	**108.2** ± 0.8
Medium	HalfCheetah	**43.8** ± 0.5	**44.2** ± 0.6
Medium	Hopper	95.4 ± 3.4	**96.6** ± 2.7
Medium	Walker2d	83.2 ± 3.5	**84.4** ± 2.6
Average		88.4	**89.1**

Adaptation Performance. In KUKA pick-and-place task, we define the guidance of the conditional diffusion model as the gradient of the reward function about the distance between the current location and the target location. Then, the adaptation performance is displayed in Table 3.

There are two setups in KUKA benchmark. In setup 1, the four blocks are initialized randomly on the floor, while in setup 2, the four blocks are stacked at a random location at the beginning. As shown in Table 3, AdaptDiffuser outperforms Diffuser greatly on both setups while achieving higher performance at setup 2 because all of the blocks start from the same horizontal position. We visualize a successful case of the KUKA pick-and-place task in Figure 5, and more visualization results can be seen in Appendix B.

5.4. Ablation Study

5.4.1. Ablation on Iterative Phases

In order to verify the lifting effect of iterative data generation of our method AdaptDiffuser to improve the performance of the planner, we conduct an ablation experiment on the number of iterative phases of AdaptDiffuser in the MuJoCo environment of D4RL.

As shown in Table 4, with "Medium" dataset, due to the low quality of the original dataset, although the data generated in the first phase has greatly supplemented the training dataset and greatly improved the performance (referring to Sec 5.2.2), the performance achieved after the second phase is still significantly improved compared with that of the first phase. However, for "Medium-Expert" dataset, because the expert data of the dataset has covered most of the environment, and the newly generated data is only more suitable for the planner to learn. So, after a certain improvement in the first phase, the subsequent growth is not obvious. The above experiments verify the effectiveness of AdaptDiffuser for the multi-phase iterative paradigm, and also show that the boosting effect is no longer obvious after the algorithm performance reaches a certain level.

Table 5. **Ablation study on different amounts of expert data.**

Amount of Data	20% $\mathcal{D}$	50% $\mathcal{D}$	100%$\mathcal{D}$
Diffuser	105.0	107.9	123.0
AdaptDiffuser	**112.5**	**123.8**	**167.9**

Table 6. **Model Size of AdaptDiffuser.**

Environment	Total Parameters (Model Size)
MuJoCo	3.96 M
Maze2D	3.68 M
KUKA Robot	64.9 M

5.4.2. Ablation on Insufficient Data & Training

To demonstrate the superiority of our method over previous diffusion-based work Diffuser (Janner et al., 2022) when the expert data is limited and the training is insufficient, we conducted experiments on the Maze2d-Large dataset using different percentages of expert data (e.g. 20%, 50%) with only 25% training steps to train our model. The results are shown in Table 5. The setting 100%$\mathcal{D}$ denotes the full training setting. We can see our AdaptDiffuser, which uses only 50% data and 25% training steps, beats the fully trained Diffuser. AdaptDiffuser can achieve good performance with a small amount of expert data and training steps.

5.4.3. Model Size and Running Time

We show the model size of AdaptDiffuser measured by the number of parameters in Table 6 here. And we also analyze the testing time and training time performance in Appendix D. From the analysis, we can see that the inference time of AdaptDiffuser is almost equal to that of Diffuser (Janner et al., 2022).

6. Conclusion

We present AdaptDiffuser, a method for improving the performance of diffusion-based planners in offline reinforcement learning through self-evolution. By generating diverse, high-quality and heterogeneous expert data using a reward-guided diffusion model and filtering out infeasible data using a rule-based discriminator, AdaptDiffuser is able to enhance the performance of diffusion models in existing decision-making tasks, especially the goal-conditioned tasks, and further improve the adaptability in unseen tasks without any expert data. Our experiments on two widely-used offline RL benchmarks and our carefully designed unseen tasks in KUKA and Maze2D environments validate the effectiveness of AdaptDiffuser.

Discussion of Limitation. Our method achieves better performance by generating high-quality synthetic data but increases the amount of computation required in training with almost no increase in inference time. Besides, although AdaptDiffuser has proven its effectiveness in several scenarios (e.g. MuJoCo, Maze2d, KUKA), it still faces challenges in high-dimensional observation space tasks. More detailed discussions are given in Appendix F.

Future Works. Further improving the sampling speed and exploring tasks with high-dimensional input are potential areas for future works. And with the help of ChatGPT (Ouyang et al., 2022), we can use prompts to directly generate diverse maze settings to assist synthetic data generation which is also a promising direction. We provide some examples in Appendix G.

Acknowledgements

This paper is partially supported by the National Key R&D Program of China No.2022ZD0161000 and the General Research Fund of Hong Kong No.17200622.

References

Agarwal, R., Schuurmans, D., and Norouzi, M. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pp. 104–114. PMLR, 2020.

Ajay, A., Du, Y., Gupta, A., Tenenbaum, J., Jaakkola, T., and Agrawal, P. Is conditional generative modeling all you need for decision-making? In *International Conference on Learning Representations*, 2023.

Argenson, A. and Dulac-Arnold, G. Model-based offline planning. In *International Conference on Learning Representations*, 2021.

Austin, J., Johnson, D. D., Ho, J., Tarlow, D., and van den Berg, R. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*, 2021.

Buckman, J., Gelada, C., and Bellemare, M. G. The importance of pessimism in fixed-dataset policy optimization. In *International Conference on Learning Representations*, 2021.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021a.

Chen, N., Zhang, Y., Zen, H., Weiss, R. J., Norouzi, M., and Chan, W. Wavegrad: Estimating gradients for waveform generation. In *International Conference on Learning Representations*, 2021b.

Dhariwal, P. and Nichol, A. Q. Diffusion models beat GANs on image synthesis. In *Advances in Neural Information Processing Systems*, 2021.

Du, Y. and Mordatch, I. Implicit generation and generalization in energy-based models. In *Advances in Neural Information Processing Systems*, 2019.

Du, Y., Li, S., and Mordatch, I. Compositional visual generation with energy based models. In *Advances in Neural Information Processing Systems*, 2020.

Fan, L., Wang, G., Jiang, Y., Mandlekar, A., Yang, Y., Zhu, H., Tang, A., Huang, D.-A., Zhu, Y., and Anandkumar, A. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

Feller, W. On the theory of stochastic processes, with particular reference to applications. In *Selected Papers I*, pp. 769–798. Springer, 2015.

Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Garrett, C. R., Lozano-Pérez, T., and Kaelbling, L. P. Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pp. 440–448, 2020.

Grathwohl, W., Wang, K.-C., Jacobsen, J.-H., Duvenaud, D., and Zemel, R. Learning the stein discrepancy for training and evaluating energy-based models without sampling. In *International Conference on Machine Learning*, 2020.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

Hyvärinen, A. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 2005.

Janner, M., Li, Q., and Levine, S. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34: 1273–1286, 2021.

Janner, M., Du, Y., Tenenbaum, J., and Levine, S. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, pp. 9902–9915. PMLR, 2022.

Kidambi, R., Rajeswaran, A., Netrapalli, P., and Joachims, T. Morel: Model-based offline reinforcement learning. *Advances in neural information processing systems*, 33: 21810–21823, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Kostrikov, I., Nair, A., and Levine, S. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33: 1179–1191, 2020.

LeCun, Y., Chopra, S., Hadsell, R., Huang, F. J., and et al. A tutorial on energy-based learning. In *Predicting Structured Data*. MIT Press, 2006.

Levine, S., Kumar, A., Tucker, G., and Fu, J. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Misra, D. Mish: A self regularized non-monotonic activation function. In *31st British Machine Vision Conference 2020, BMVC 2020, Virtual Event, UK, September 7-10, 2020*, 2020.

Nijkamp, E., Hill, M., Zhu, S.-C., and Wu, Y. N. Learning non-convergent non-persistent short-run MCMC toward energy-based model. In *Advances in Neural Information Processing Systems*, 2019.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

Prudencio, R. F., Maximo, M. R., and Colombini, E. L. A survey on offline reinforcement learning: Taxonomy, review, and open problems. *arXiv preprint arXiv:2203.01387*, 2022.

Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., and Chen, M. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, 2022.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted*

Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18, pp. 234–241. Springer, 2015.

Schreiber, G., Stemmer, A., and Bischoff, R. The fast research interface for the kuka lightweight robot. In *IEEE workshop on innovative robot control architectures for demanding (Research) applications how to modify and enhance commercial controllers (ICRA 2010)*, pp. 15–21. Citeseer, 2010.

Siegel, N., Springenberg, J. T., Berkenkamp, F., Abdolmaleki, A., Neunert, M., Lampe, T., Hafner, R., Heess, N., and Riedmiller, M. Keep doing what worked: Behavior modelling priors for offline reinforcement learning. In *International Conference on Learning Representations*, 2020.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pp. 2256–2265. PMLR, 2015.

Song, J., Meng, C., and Ermon, S. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021.

Song, Y. and Ermon, S. Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems*, 2019.

Sutton, R. S. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033. IEEE, 2012.

Wang, K., Zhao, H., Luo, X., Ren, K., Zhang, W., and Li, D. Bootstrapped transformer for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2022.

Wang, Z., Hunt, J. J., and Zhou, M. Diffusion policies as an expressive policy class for offline reinforcement learning. In *International Conference on Learning Representations*, 2023.

Williams, G., Aldrich, A., and Theodorou, E. Model predictive path integral control using covariance variable importance sampling. *arXiv preprint arXiv:1509.01149*, 2015.

Wu, Y. and He, K. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

Wu, Y., Tucker, G., and Nachum, O. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

Yang, S., Wang, Z., Zheng, H., Feng, Y., and Zhou, M. A regularized implicit policy for offline reinforcement learning. *arXiv preprint arXiv:2202.09673*, 2022.

Yu, T., Thomas, G., Yu, L., Ermon, S., Zou, J. Y., Levine, S., Finn, C., and Ma, T. Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems*, 33:14129–14142, 2020.

Zhang, L. Dirac delta function of matrix argument. *International Journal of Theoretical Physics*, 60(7):2445–2472, 2021.

Zhou, L., Du, Y., and Wu, J. 3D shape generation and completion through point-voxel diffusion. In *International Conference on Computer Vision*, 2021.

A. Classifier-Guided Diffusion Model for Planning

In this section, we introduce theoretical analysis of conditional diffusion model in detail. We start with an unconditional diffusion probabilistic model with a standard reverse process as $p_\theta(\tau^i|\tau^{i+1})$. Then, with a specific label y (for example, goal point in Maze2D or specific reward function in MuJoCo) which is to be conditioned on given a noised trajectory τ^i, the reverse diffusion process can be redefined as $p_{\theta,\phi}(\tau^i|\tau^{i+1}, y)$. Apart from the parameters θ of original diffusion model, a new parameter ϕ is introduced here which describes the probability transfer model from noisy trajectory τ^i to the specific label y which is denoted as $p_\phi(y \mid \tau^i)$.

Lemma A.1. *The marginal probability of a conditional Markov's noising process q conditioned on y is equal to the marginal probability of the unconditional noising process.*

$$q\left(\tau^{i+1} \mid \tau^i\right) = q\left(\tau^{i+1} \mid \tau^i, y\right) \tag{12}$$

Proof.

$$\begin{aligned} q\left(\tau^{i+1} \mid \tau^i\right) &= \int_y q\left(\tau^{i+1}, y \mid \tau^i\right) dy \\ &= \int_y q\left(\tau^{i+1} \mid \tau^i, y\right) p_\phi\left(y \mid \tau^i\right) dy \\ &= q\left(\tau^{i+1} \mid \tau^i, y\right) \int_y p_\phi\left(y \mid \tau^i\right) dy \\ &= q\left(\tau^{i+1} \mid \tau^i, y\right) \end{aligned}$$

The third line holds because $q\left(\tau^{i+1} \mid \tau^i, y\right)$ fits another y-independent transition probability according to its definition. □

Lemma A.2. *The probability distribution of specific label y conditioned on τ^i does not depend on τ^{i+1}.*

$$p_{\theta,\phi}\left(y \mid \tau^i, \tau^{i+1}\right) = p_\phi\left(y \mid \tau^i\right) \tag{13}$$

Proof.

$$\begin{aligned} p_{\theta,\phi}\left(y \mid \tau^i, \tau^{i+1}\right) &= q\left(\tau^{i+1} \mid \tau^i, y\right) \frac{p_\phi\left(y \mid \tau^i\right)}{q\left(\tau^{i+1} \mid \tau^i\right)} \\ &= q\left(\tau^{i+1} \mid \tau^i\right) \frac{p_\phi\left(y \mid \tau^i\right)}{q\left(\tau^{i+1} \mid \tau^i\right)} \\ &= p_\phi\left(y \mid \tau^i\right) \end{aligned}$$

□

Theorem A.3. *The conditional sampling probability $p_{\theta,\phi}(\tau^i \mid \tau^{i+1}, y)$ is proportional to unconditional transition probability $p_\theta(\tau^i \mid \tau^{i+1})$ multiplied by classified probability $p_\phi(y \mid \tau^i)$.*

$$p_{\theta,\phi}(\tau^i \mid \tau^{i+1}, y) = Z p_\theta(\tau^i \mid \tau^{i+1}) p_\phi(y \mid \tau^i) \tag{14}$$

Proof.

$$\begin{aligned} p_{\theta,\phi}(\tau^i \mid \tau^{i+1}, y) &= \frac{p_{\theta,\phi}\left(\tau^i, \tau^{i+1}, y\right)}{p_{\theta,\phi}\left(\tau^{i+1}, y\right)} \\ &= \frac{p_{\theta,\phi}\left(\tau^i, \tau^{i+1}, y\right)}{p_\phi\left(y \mid \tau^{i+1}\right) p_\theta\left(\tau^{i+1}\right)} \\ &= \frac{p_\theta\left(\tau^i \mid \tau^{i+1}\right) p_{\theta,\phi}\left(y \mid \tau^i, \tau^{i+1}\right) p_\theta\left(\tau^{i+1}\right)}{p_\phi\left(y \mid \tau^{i+1}\right) p_\theta\left(\tau^{i+1}\right)} \\ &= \frac{p_\theta\left(\tau^i \mid \tau^{i+1}\right) p_{\theta,\phi}\left(y \mid \tau^i, \tau^{i+1}\right)}{p_\phi\left(y \mid \tau^{i+1}\right)} \\ &= \frac{p_\theta\left(\tau^i \mid \tau^{i+1}\right) p_\phi\left(y \mid \tau^i\right)}{p_\phi\left(y \mid \tau^{i+1}\right)} \end{aligned} \tag{15}$$

The term $p_\phi\left(y \mid \tau^{i+1}\right)$ can be seen as a constant since it's not conditioned on τ^i at the diffusion timestep i. □

Although exact sampling from this distribution (Equation 14) is difficult, (Sohl-Dickstein et al., 2015) demonstrates that it can be approximated as a modified Gaussian distribution. We show the derivation here.

On one hand, as Equation 4 shows, we can formulate the denoising process with a Gaussian distribution:

$$p_\theta(\tau^i \mid \tau^{i+1}) = \mathcal{N}(\mu, \Sigma) \tag{16}$$

$$\log p_\theta(\tau^i \mid \tau^{i+1}) = -\frac{1}{2}(\tau^i - \mu)^T \Sigma^{-1}(\tau^i - \mu) + C \tag{17}$$

And on the other hand, the number of diffusion steps are usually large, so the difference between τ^i and τ^{i+1} is small enough. We can apply Taylor expansion around $\tau^i = \mu$ to $\log p_\phi(y \mid \tau^i)$ as,

$$\log p_\phi\left(y \mid \tau^i\right) = \log p_\phi\left(y \mid \tau^i\right)|_{\tau^i=\mu} + \left(\tau^i - \mu\right) \nabla_{\tau^i} \log p_\phi\left(y \mid \tau^i\right)\Big|_{\tau^i=\mu} \tag{18}$$

Therefore, synthesize Equation 17 and 18, we derive,

$$\begin{aligned}
\log p_{\theta,\phi}(\tau^i|\tau^{i+1}, y) &= \log p_\theta(\tau^i|\tau^{i+1}) + \log p_\phi(y|\tau^i) + C_1 \\
&= -\frac{1}{2}\left(\tau^i - \mu\right)^T \Sigma^{-1}\left(\tau^i - \mu\right) + \left(\tau^i - \mu\right) \nabla \log p_\phi\left(y \mid \tau^i\right) + C_2 \\
&= -\frac{1}{2}\left(\tau^i - \mu - \Sigma\nabla \log p_\phi\left(y \mid \tau^i\right)\right)^T \Sigma^{-1}\left(\tau^i - \mu - \Sigma\nabla \log p_\phi\left(y \mid \tau^i\right)\right) + C_3
\end{aligned} \tag{19}$$

which means,

$$p_{\theta,\phi}(\tau^i|\tau^{i+1}, y) \approx \mathcal{N}(\tau^i; \mu + \Sigma\nabla_\tau \log p_\phi\left(y \mid \tau^i\right), \Sigma) \tag{20}$$

And it's equal to Equation 9. Proven.

B. Visualization Results of KUKA Pick-and-Place Task

In this section, we show more visualization results about KUKA pick-and-place task. We require the KUKA Robot Arm to pick green, yellow, blue and red blocks with random initialized positions on the right side of the table one by one and move them to the left side in the order of yellow, blue, green and red (from near to far).

B.1. Pick and Place 1st Green Block

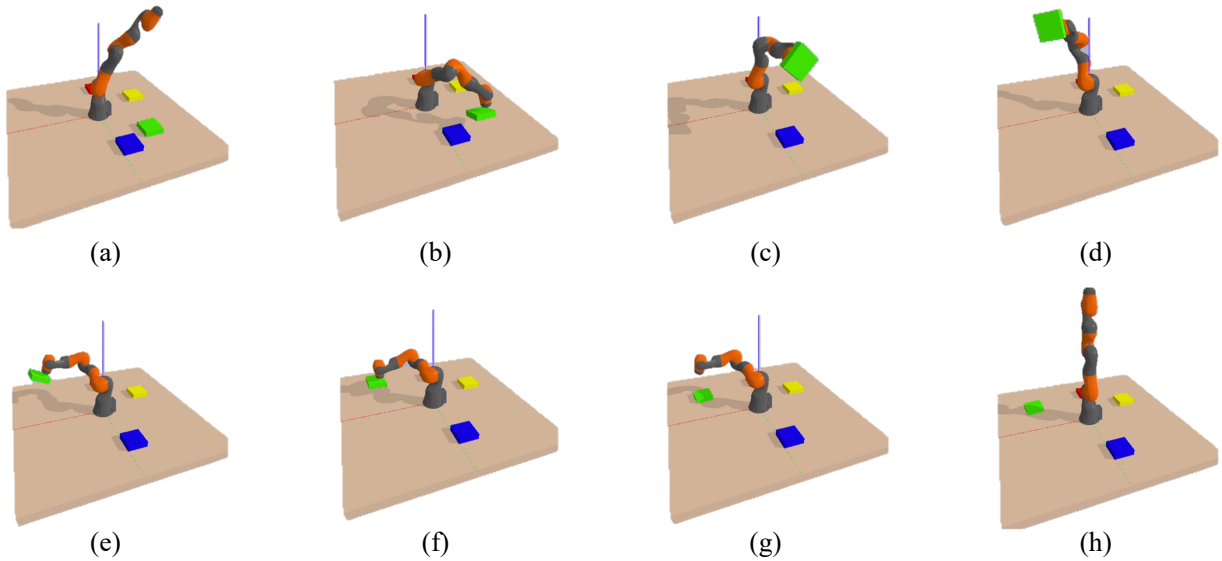


Figure 6. The Process of Pick and Place Block 1 (Green Block)

B.2. Pick and Place 2nd Yellow Block

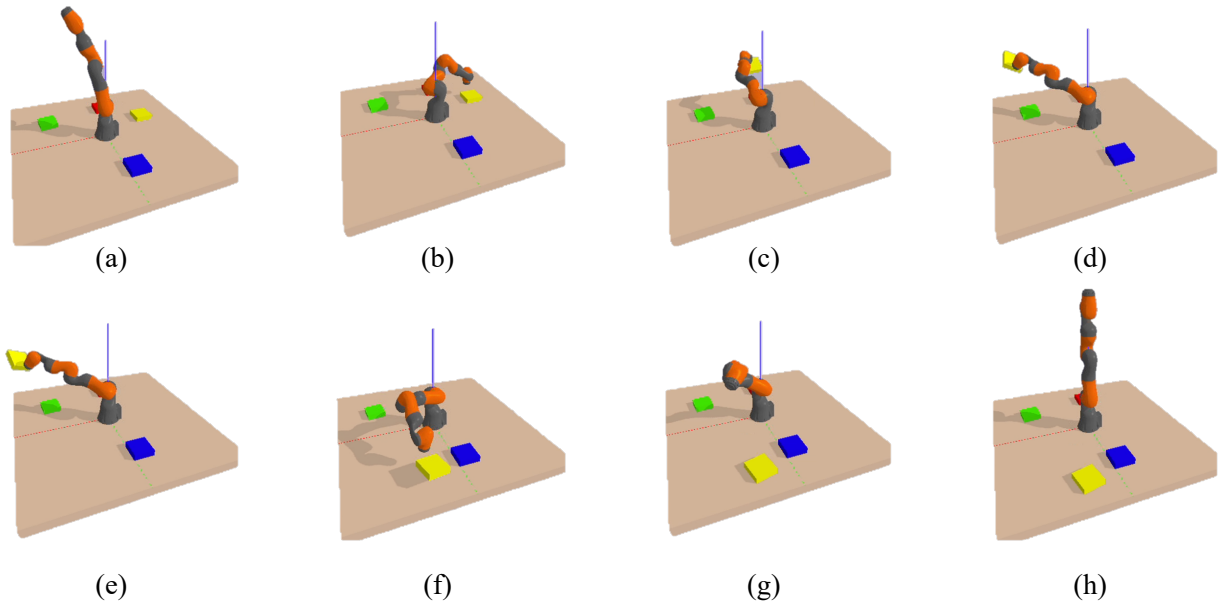


Figure 7. The Process of Pick and Place Block 2 (Yellow Block)

B.3. Pick and Place 3rd Blue Block

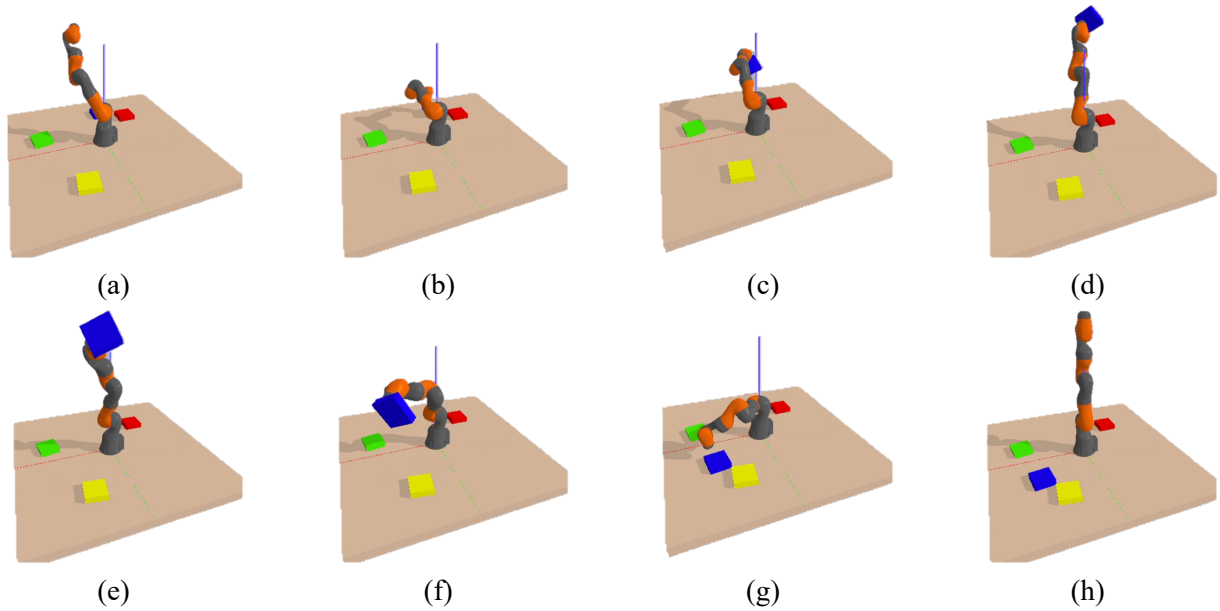


Figure 8. The Process of Pick and Place Block 3 (Blue Block)

B.4. Pick and Place 4th Red Block

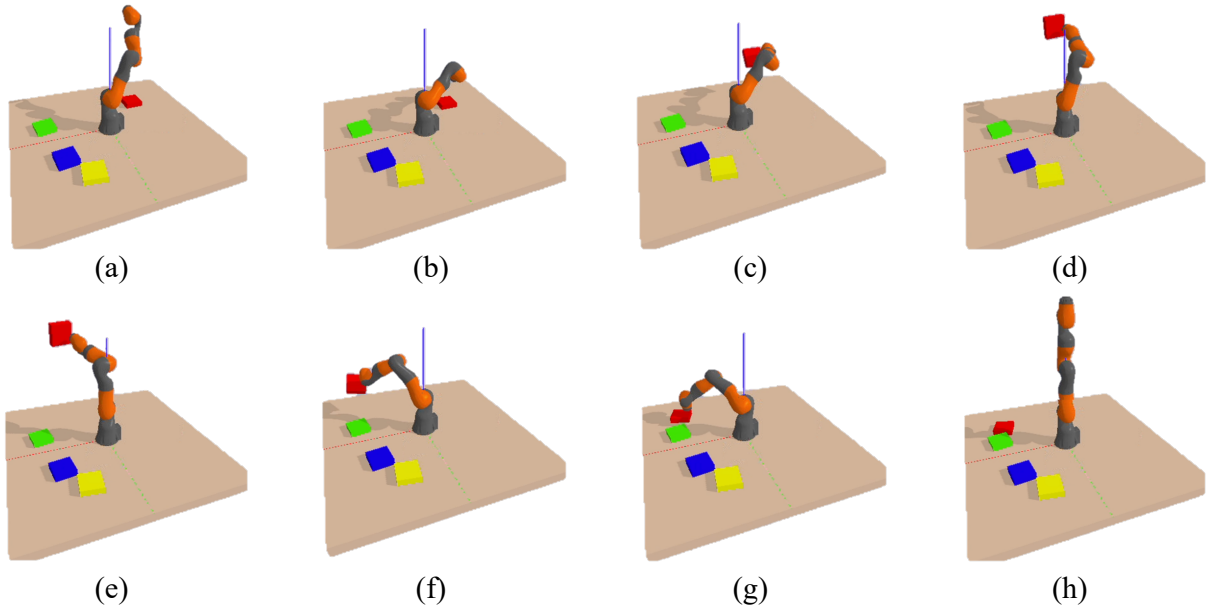


Figure 9. The Process of Pick and Place Block 4 (Red Block)

C. Implementation Details and Hyperparameters

C.1. Details of Baseline Performances

Maze2D Tasks. We perform two different tasks on the Maze2D environment to validate the performance enhancement and adaptation ability of AdaptDiffuser on seen and unseen tasks.

- **Overall Performance of Navigation Task**: We report the performance of CQL and IQL on the standard Maze2D environments from Table 2 in D4RL whitepaper (Fu et al., 2020) and follow the hyperparameter settings described in (Janner et al., 2022). The performance of Diffuser also refers to Table 1 in (Janner et al., 2022). To reproduce the experimental results, we use the official implementation from the authors of IQL[1] and Diffuser[2].
- **Navigation with Gold Coin Picking Task**: We modified the official code of Diffuser and tuned over the hyperparameter $\alpha \in \{-50, -100, -200\}$ (the scalar of the guidance) in Equation 8 to adjust the planner to be competent for newly designed gold coin picking task, which is also the basis of our method AdaptDiffuser.

KUKA Pick and Place Tasks. Similar to the unseen tasks in Maze2D environment, we also ran the official implementation of IQL and Diffuser.

MuJoCo Locomotion Tasks. We report the scores of BC, CQL and IQL from Table 1 in (Kostrikov et al., 2022). We take down scores of DT from Table 2 in (Chen et al., 2021a), TT from Table 1 in (Janner et al., 2021), MOPO from Table 1 in (Yu et al., 2020), MOReL from Table 2 in (Kidambi et al., 2020), MBOP from Table 1 in (Argenson & Dulac-Arnold, 2021) and Diffuser from Table 2 in (Janner et al., 2022). All baselines are trained using the same offline dataset collected by a specific expert policy.

Table 7. **Metric Values for Reward Discriminator in MuJoCo Environment.** The rewards are calculated utilizing D4RL (Fu et al., 2020) locomotion suite.

Dataset	Environment	1st Phase	2nd Phase
Med-Expert	HalfCheetah	10840	10867
Med-Expert	Hopper	3639	3681
Med-Expert	Walker2d	4900	4950
Medium	HalfCheetah	5005	5150
Medium	Hopper	3211	3225
Medium	Walker2d	3700	3843
Med-Replay	HalfCheetah	4600	4800
Med-Replay	Hopper	3100	3136
Med-Replay	Walker2d	3900	3920

C.2. Metric Values for Reward Discriminator

Maze2D Environment. For the three different-size Maze2D settings, unlike MuJoCo, different trajectories are different in lengths which achieve different rewards. So, we not only consider the absolute value of the rewards $\mathcal{R}$ but also introduce trajectory length $\mathcal{L}$ and reward-length ratio into the criteria of discrimination. We prefer trajectories with longer lengths or those having higher reward-length ratios. Additionally, we denote the maximum episode steps of the environment as Max_e (Maze2D-UMaze: 300, Maze2D-Medium: 600, Maze2D-Large: 800). And then, we have following metrics to filter out high-quality data.

- **Maze2D-UMaze**: The trajectory is required to satisfy $\mathcal{L} > 200$ or $\mathcal{L} > 50$ and $\mathcal{R} + 1.0 * (Max_e - \mathcal{L}) > 210$ which is equal to measure the $\mathcal{R}/\mathcal{L}$.
- **Maze2D-Medium**: The trajectory is required to satisfy $\mathcal{L} > 450$ or $\mathcal{L} > 200$ and $\mathcal{R} + 1.0 * (Max_e - \mathcal{L}) > 400$.
- **Maze2D-Large**: The trajectory is required to satisfy $\mathcal{L} > 650$ or $\mathcal{L} > 270$ and $\mathcal{R} + 1.0 * (Max_e - \mathcal{L}) > 400$.

[1] https://github.com/ikostrikov/implicit_q_learning
[2] https://github.com/jannerm/diffuser

KUKA Robot Arm. For the KUKA Robot Arm environment, we define a sparse reward function that achieves one if and only if the placement is successful and zero otherwise. Therefore, we take the condition $\mathcal{R} >= 2.0$ which means at least half of the four placements are successful.

MuJoCo Environment. For MuJoCo locomotion environment, as we describe in Sec. 5.2.2, we directly use the reward derived after generated state sequence and action sequence to filter out high-quality synthetic data. The specific values for MuJoCo are shown in Table 7.

C.3. Amount of Synthetic Data for Each Iteration

The amount of synthetic data for each iteration is another important hyperparameter for AdaptDiffuser. Different tasks have different settings. We give detailed hyperparameters here.

Table 8. **Amount of Synthetic Data for Each Iteration.** The number of synthetic data for KUKA Arm pick-and place task consists of 1000 generated trajectories and 10000 cross-domain trajectories from the unconditional stacking task.

Dataset	Task	# of Expert Data	# of Synthetic Data
MuJoCo	Locomotion	10^6, 2×10^6	50000
Maze2D	Navigation	10^6, 2×10^6, 4×10^6	10^6
Maze2D	Gold Coin Picking	0	10^6
KUKA Robot	Unconditional Stacking	10000	-
KUKA Robot	Pick-and-Place	0	11000

C.4. Other Details

1. A temporal U-Net (Ronneberger et al., 2015) with 6 repeated residual blocks is employed to model the noise ϵ_θ of the diffusion process. Each block is comprised of two temporal convolutions, each followed by group norm (Wu & He, 2018), and a final Mish non-linearity (Misra, 2020). Timestep embeddings are generated by a single fully-connected layer and added to the activation output after the first temporal convolution of each block.

2. The diffusion model is trained using the Adam optimizer (Kingma & Ba, 2015) with a learning rate of 2×10^{-4} and batch size of 32.

3. The training steps of the diffusion model are $1M$ for MuJoCo locomotion task, $2M$ for tasks on Maze2D and $0.7M$ for KUKA Robot Arm tasks.

4. The planning horizon T is set as 32 in all locomotion tasks, 128 for KUKA pick-and-place, 128 in Maze2D-UMaze, 192 in Maze2D-Medium, and 384 in Maze2D-Large.

5. We use $K = 100$ diffusion steps for all locomotion tasks, 1000 for KUKA robot arm tasks, 64 for Maze2D-UMaze, 128 for Maze2D-Medium, and 256 for Maze2D-Large.

6. We choose 2-norm as the auxiliary guided function in the combination setting of Section 4.3 and the guidance scale $\alpha \in \{1, 5, 10, 50, 100\}$ of which the exact choice depends on the specific task.

D. Testing-time and Training-time Analysis

D.1. Testing-time Characteristic of AdaptDiffuser

AdaptDiffuser only generates synthetic data during training and performs denoising once during inference to obtain the optimal trajectory. We show the inference time of generating an action taken by Diffuser (Janner et al., 2022) and our method in Table 9 and Table 10. All these data are tested with one *NVIDIA RTX 3090 GPU*.

Table 9. **Testing Time in D4RL MuJoCo Environment.** The unit in the table is second (s).

Dataset	Environment	Diffuser	AdaptDiffuser
Med-Expert	HalfCheetah	1.38 s	1.41 s
Med-Expert	Hopper	1.57 s	1.59 s
Med-Expert	Walker2d	1.60 s	1.56 s
Medium	HalfCheetah	1.40 s	1.40 s
Medium	Hopper	1.60 s	1.56 s
Medium	Walker2d	1.57 s	1.57 s
Med-Replay	HalfCheetah	1.43 s	1.37 s
Med-Replay	Hopper	1.59 s	1.55 s
Med-Replay	Walker2d	1.55 s	1.58 s

Table 10. **Testing Time in D4RL Maze2D and KUKA Environments.** The test time of KUKA is derived by dividing the trajectory generation time by horizon size. The unit in the table is second (s).

Environment	Diffuser	AdaptDiffuser
Maze2D U-Maze	0.70 s	0.69 s
Maze2D Medium	1.42 s	1.44 s
Maze2D Large	2.80 s	2.76 s
KUKA Pick and Place	0.21 s	0.21 s

From the tables, we can see that the inference time of AdaptDiffuser is almost equal to that of Diffuser (Janner et al., 2022). And because the denoising steps of different datasets are different, the testing times are different between environments. For MuJoCo, the inference time of an action is approximately 1.5s, while for Maze2D the inference time is about 1.6s (on average of three environments), and for KUKA about 0.21s. The inference time is feasible for real-time robot control. Additionally, in Section 5.4.2 of our paper, we have also demonstrated how limited number of high quality expert data would affect our method's performance.

What's more, as suggested in Diffuser (Janner et al., 2022), we can improve the testing time by warm-starting the state diffusion, which means we start with the state sequence generated from the previous environment step and then reduce the number of denoising steps.

Table 11. **Synthetic Data Generation Time and Training Time in MuJoCo Environment.** The synthetic data generation time listed here is about the time to generate one high-quality trajectory. The total training time of AdaptDiffuser is the sum of the following three parts. The quality standard of selected trajectories are the same as those stated in Appendix C.2. The unit in the table is hour (h).

Dataset	Environment	Synthetic Data Gen. Time	AdaptDiffuser Fine-Tuning	Diffuser Training
Med-Expert	HalfCheetah	4.4 h	6.8 h	44.2 h
Med-Expert	Hopper	5.7 h	6.4 h	37.0 h
Med-Expert	Walker2d	3.0 h	6.6 h	43.0 h
Medium	HalfCheetah	2.4 h	7.0 h	45.3 h
Medium	Hopper	4.8 h	6.2 h	36.2 h
Medium	Walker2d	4.7 h	6.4 h	43.0 h
Med-Replay	HalfCheetah	15.7 h	7.4 h	45.3 h
Med-Replay	Hopper	11.9 h	6.5 h	36.1 h
Med-Replay	Walker2d	4.3 h	6.4 h	42.8 h

D.2. Training-time Characteristic of AdaptDiffuser

The training time of AdaptDiffuser can be seen as the sum of synthetic data generation time and diffusion model training time. The synthetic data generation time depends on the quality standard of the trajectory to be selected.

What's more, to accelerate the training, we use the warming-up technique which takes the pre-trained Diffuser model as the basis of AdaptDiffuser, and then performs fine-tuning on new generated data with fewer training steps (1/4 in actual use). Then we show these three parts' times in Table 11. All these times are tested with one *NVIDIA RTX 3090 GPU*.

It can be found from the table that the model training time dominates the total pre-training time while the extra time spent, such as synthetic data generation, is a relatively small part. The total time required to pre-train AdaptDiffuser is on average 54 hours (sum of the three parts) comparable to Diffuser's 41 hours.

Besides, the data generation process can be executed parallel. For example, in our D4RL MuJoCo environment, we generate 10 trajectories for each dataset at each phase. Under parallel settings, the total time to collect all ten synthetic trajectories is the same as the time to collect one trajectory. If using more GPUs, the synthetic data generation time can be further reduced.

E. Comparison with Decision Diffuser

Decision Diffuser (DD) (Ajay et al., 2023) is a concurrent work with ours and improves the performance of Diffuser (Janner et al., 2022) by introducing planning with classifier-free guidance and acting with inverse-dynamics.

Generally speaking, our method is a general algorithm that enables diffusion-based planners to have self-evolving ability that can perform well on existing and unseen (zero-shot) tasks, mainly by generating high-quality synthetic data with reward and dynamics consistency guidance for diverse tasks simultaneously. Therefore, regardless of which diffusion-based planner to be used, there can exist AdaptDiffuser, AdaptDecisionDiffuser, etc. It means that the method we introduce to make the planner self-evolving does not conflict with the improvements proposed by Decision Diffuser. The improvements of these two works can complement each other to further enhance the performance of diffusion model-based planners.

We also compare the performance of Decision Transformer (DT) (Chen et al., 2021a), Trajectory Transformer (TT) (Janner et al., 2021), Diffuser (Janner et al., 2022), Decision Diffuser (Ajay et al., 2023) and our method here. Results about Decision Diffuser are quoted from (Ajay et al., 2023).

Table 12. **Performance Comparison with Decision Diffuser in MuJoCo Environment.** We report normalized average returns of D4RL tasks (Fu et al., 2020) in the table. And the mean and the standard error are calculated over 3 random seeds.

Dataset	Environment	DT	TT	Diffuser	Decision Diffuser	AdaptDiffuser
Med-Expert	HalfCheetah	86.8	**95.0**	88.9	90.6	89.6 ± 0.8
Med-Expert	Hopper	107.6	110.0	103.3	**111.8**	**111.6** ± 2.0
Med-Expert	Walker2d	**108.1**	101.9	106.9	**108.8**	**108.2** ± 0.8
Medium	HalfCheetah	42.6	46.9	42.8	**49.1**	44.2 ± 0.6
Medium	Hopper	67.6	61.1	74.3	79.3	**96.6** ± 2.7
Medium	Walker2d	74.0	79.0	79.6	82.5	**84.4** ± 2.6
Med-Replay	HalfCheetah	36.6	41.9	37.7	**39.3**	**38.3** ± 0.9
Med-Replay	Hopper	82.7	91.5	93.6	**100.0**	92.2 ± 1.5
Med-Replay	Walker2d	66.6	82.6	70.6	75.0	**84.7** ± 3.1
Average		74.7	78.9	77.5	81.8	**83.4**

From the table, we can see that in most datasets, the performance of AdaptDiffuser is comparable to or better than that of Decision Diffuser. And the normalized average return of AdaptDiffuser is 83.4 higher than all of the other methods (i.e. 74.7 of DT, 78.9 of TT, 77.5 of Diffuser and 81.8 of Decision Diffuser).

F. Discussions

F.1. Adapt AdaptDiffuser to Maze2D Gold Coin Picking Task with Coin Locating Far from the Optimal Path

AdaptDiffuser works when the gold coin is located nowhere near the optimal path. Figure 4 of our paper has shown one case. The sub-figure (b) of Figure 4 show the optimal path when there are no gold coins in the maze. (The generated route walks at the bottom of the figure.) And then if we add a gold coin in the (4,2) position of the maze, AdaptDiffuser will generate a new path that passes through the gold coin as shown in the sub-figure (d) of Figure 4. (The generated route walks in the middle of the figure.)

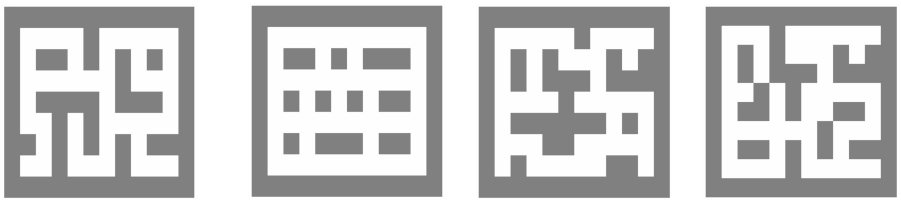
In our point of view, our method works mainly because we change the start point and goal point multiple times during training. Diffusion model can generate trajectories that have not been seen in the expert dataset. And as long as the paths generated during training can cover the entire trajectory space as much as possible, AdaptDiffuser can generate the path through any location of the gold coin during planning. However, it is true that the success rate of generating trajectories for some extremely hard cases that the gold coin is far from the planned path and the agent has to take a turn back to obtain the gold coin, is lower than that of common cases.

F.2. Adapt AdaptDiffuser to High-dimensional Observation Space Tasks

AdaptDiffuser is feasible for high-dimensional observation space tasks. One possible and widely-used solution, we suggest, is to add an embedding module (e.g. MLP) after input to convert the data from high-dimensional space to latent space, and then employ AdaptDiffuser in latent space to solve the problem. Stable Diffusion (Rombach et al., 2022) has shown the effectiveness of this method, which deploys an Auto-Encoder to encode image into a latent representation and uses a decoder to reconstruct the image from the latent after denoising. MineDoJo (Fan et al., 2022) also takes this technique and achieves outstanding performance in image-based RL domain.

G. Generate Diverse Maze Layouts with ChatGPT

Inspired by the remarkable generation capabilities demonstrated by recent advancements in large language models (LLMs), exemplified by ChatGPT, we propose a novel approach that harnesses the potential of LLM to accelerate the process of synthetic data generation. In this section, we focus specifically on utilizing LLM to assist in generating diverse Maze layouts. This objective is driven by the need to create a multitude of distinct maze layouts to facilitate varied path generations, ultimately enhancing the performance and adaptability of AdaptDiffuser. Traditionally, the manual design of feasible and terrain complex maze environments is a time-consuming endeavor that requires to try and adjust multiple times. In light of this challenge, leveraging ChatGPT for maze environment generation emerges as an appealing alternative, streamlining the process and offering enticing advantages. We show the generated examples in Fig. 10. Besides, we can ask the ChatGPT to summarize the rules of generating feasible mazes, shown in Fig. 11.



(a) Existing Maze (b) Generated Mazes with ChatGPT

Figure 10. **Generated Maze examples by ChatGPT.** From simple terrain to complex terrain (with multiple dead ends and loops).

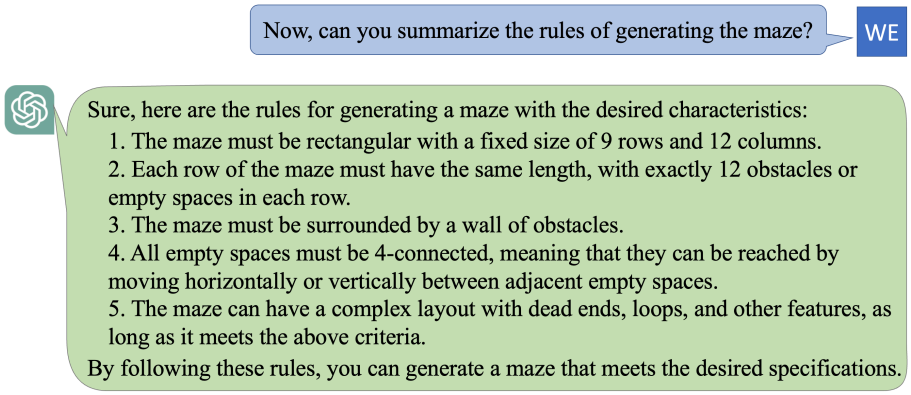


Figure 11. **Rules for generating maze layouts summarized by ChatGPT.**

We also give our prompts here. We find that providing ChatGPT with a few existing feasible maze examples (few-shot) can effectively improve the quality of the generated mazes, so we design the prompts in this way. From prompt 1 to prompt 2, we also find that the terrains of generated mazes are exactly from simple to complex.

Prompt1: "I will give you a legal string expression of a MAZE. In the MAZE, the '#' represents the obstacles and the 'O' represents the empty space. Could you generate one more maze with different terrain obeying to the rules: The MAZE should be 9*12, and the surrounding of the MAZE should be obstacles, that is '#', and all empty places should be 4-connected. The example maze is

```
LARGE_MAZE = "############\\"+
             "#OOOO#OOOOO#\\"+
             "#O##O#O#O#O#\\"+
             "#OOOOOO#OOO#\\"+
             "#O####O###O#\\"+
             "#OO#O#OOOOO#\\"+
             "##O#O#O#O###\\"+
             "#OO#OOO#OOO#\\"+
             "############" "
```

Prompt2: "Please generate more complex Maze that has more complex terrains (i.e. more dead ends, loops, and obstacles)".