IS CONDITIONAL GENERATIVE MODELING ALL YOU NEED FOR DECISION-MAKING?

Anurag Ajay[* †‡], **Yilun Du** [*‡], **Abhi Gupta**[*‡], **Joshua Tenenbaum**[‡], **Tommi Jaakkola**[‡], **Pulkit Agrawal**[†‡]
Improbable AI Lab[†]
Massachusetts Institute Technology[‡]

ABSTRACT

Recent improvements in conditional generative modeling have made it possible to generate high-quality images from language descriptions alone. We investigate whether these methods can directly address the problem of sequential decision-making. We view decision-making not through the lens of reinforcement learning (RL), but rather through conditional generative modeling. To our surprise, we find that our formulation leads to policies that can outperform existing offline RL approaches across standard benchmarks. By modeling a policy as a return-conditional generative model, we avoid the need for dynamic programming and subsequently eliminate many of the complexities that come with traditional offline RL. We further demonstrate the advantages of modeling policies as conditional generative models by considering two other conditioning variables: constraints and skills. Conditioning on a single constraint or skill during training leads to behaviors at test-time that can satisfy several constraints together or demonstrate a composition of skills. Our results illustrate that conditional generative modeling is a powerful tool for decision-making.

1 INTRODUCTION

Over the last few years, there have been incredible advances in the field of conditional generative modeling (Ramesh et al., 2022; Jumper et al., 2021; Brown et al., 2020; Saharia et al., 2022; Lewkowycz et al., 2022). Image models like DALL-E (Ramesh et al., 2022) and ImageGen (Saharia et al., 2022) can generate accurate high-resolution images from text descriptions alone. Language models like GPT (Brown et al., 2020) can generate entire paragraphs and stories from short text prompts. More recent models like Minerva (Lewkowycz et al., 2022) can generate complete, step-by-step solutions to high-school math questions. The success of generative models in countless domains motivates us to apply them to decision-making.

Directly applying generative modeling to decision-making is not so straightforward because data is readily available from the internet in the former, but must be actively gathered from the environment in the latter. Conveniently, there exists a wide body of work that has looked into recovering high-performing policies from data logged by already operational systems (Kostrikov et al., 2022; Kumar et al., 2020; Walke et al., 2022). This is particularly useful in real-world settings where interacting with the environment is not always possible, and exploratory decisions can have fatal consequences (Dulac-Arnold et al., 2021). With access to such offline datasets, the problem of decision-making becomes similar to those where generative models have already found success.

Prior work in offline decision-making has primarily relied on tools from reinforcement learning (RL) (Kumar et al., 2020; Kostrikov et al., 2022; Wu et al., 2019; Kostrikov et al., 2021; Dadashi et al., 2021; Ajay et al., 2020; Ghosh et al., 2022). Although these ideas have found some success (Yala et al., 2022; Jang et al., 2022; Mirhoseini et al., 2020), they are not without their flaws. Since they operate in the same spirit as temporal difference (TD) learning, they are prone to many instabilities that come with using function approximation, off-policy learning, and bootstrapping together, otherwise known as the *deadly triad* (Sutton & Barto, 2018; Van Hasselt et al., 2018). Since they must also operate with a limited amount of data, they rely on tricks and heuristics to keep the policy within the distribution of the dataset. These challenges make it difficult to scale existing offline RL algorithms.

[*] denotes equal contribution. Correspondence to aajay@mit.edu, yilundu@mit.edu, abhig@mit.edu

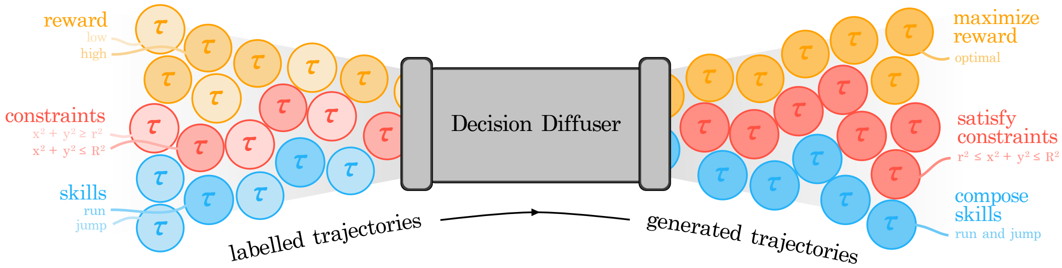

Figure 1: **Decision Making using Conditional Generative Modeling.** Framing decision making as a conditional generative modeling problem allows us to maximize rewards, satisfy constraints and compose skills.

In this paper, we formulate the problem of offline decision-making from the perspective of conditional generative modeling. Given a fixed dataset of reward-labeled trajectories, we adapt diffusion models (Sohl-Dickstein et al., 2015) to learn a return-conditional model of the trajectory. During inference, we use *classifier-free guidance* with *low-temperature sampling* to capture the best behaviors in the dataset and glean return maximizing trajectories. Our straightforward conditional generative modeling formulation outperforms existing approaches on standard D4RL tasks (Fu et al., 2020).

Viewing offline decision-making through the lens of conditional generative modeling allows going beyond conditioning on returns (Figure 1). Consider an example (detailed in Appendix A) where a robot with linear dynamics navigates an environment containing two concentric circles (Figure 2). We are given a dataset of state-action trajectories of the robot, each satisfying one of two constraints: (i) the final position of the robot is within the larger circle, and (ii) the final position of the robot is outside the smaller circle. With conditional diffusion modeling, we can use the datasets to learn a constraint-conditioned model that can generate trajectories satisfying any set of constraints. During inference, the learned trajectory model can merge constraints from the dataset and generate trajectories that satisfy the combined constraint. Figure 2 shows that the constraint-conditioned model can generate trajectories such that the final position of the robot lies between the concentric circles.

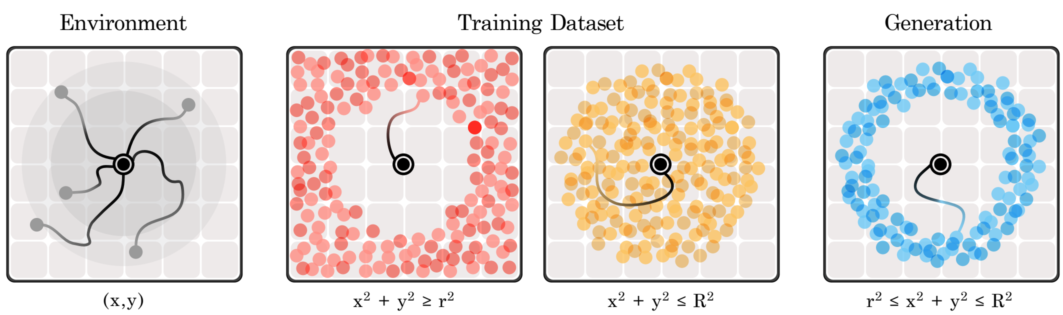

Figure 2: **Illustrative example.** We visualize the 2d robot navigation environment and the constraints satisfied by the trajectories in the dataset derived from the environment. We show the ability of the conditional diffusion model to generate trajectories that satisfy the combined constraints.

Here, we demonstrate the benefits of modeling policies as conditional generative models. First, conditioning on constraints allows policies to not only generate behaviors satisfying individual constraints but also generate novel behaviors by flexibly combining constraints at test time. Further, conditioning on skills allows policies to not only imitate individual skills but also generate novel behaviors by composing those skills. We instantiate this idea with a state-sequence based diffusion probabilistic model (Ho et al., 2020) called *Decision Diffuser*, visualized in Figure 1. In summary, our contributions include **(i)** illustrating conditional generative modeling as an effective tool in offline decision making, **(ii)** using classifier-free guidance with low-temperature sampling, instead of dynamic programming, to get return-maximizing trajectories and, **(iii)** leveraging the framework of conditional generative modeling to combine constraints and compose skills during inference flexibly.

2 BACKGROUND

2.1 REINFORCEMENT LEARNING

We formulate the sequential decision-making problem as a discounted Markov Decision Process (MDP) defined by the tuple $\langle \rho_0, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where ρ_0 is the initial state distribution, $\mathcal{S}$ and $\mathcal{A}$ are state and action spaces, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is the transition function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ gives the reward at any transition and $\gamma \in [0, 1)$ is a discount factor (Puterman, 1995). The agent acts with a stochastic policy $\pi : \mathcal{S} \rightarrow \Delta_{\mathcal{A}}$, generating a sequence of state-action-reward transitions or trajectory $\tau := (s_k, a_k, r_k)_{k \geq 0}$ with probability $p_\pi(\tau)$ and return $R(\tau) := \sum_{k \geq 0} \gamma^k r_k$. The standard objective in RL is to find a return-maximizing policy $\pi^* = \arg\max_\pi \mathbb{E}_{\tau \sim p_\pi}[R(\tau)]$.

Temporal Difference Learning TD methods (Fujimoto et al., 2018; Lillicrap et al., 2015) estimate $Q^*(s, a) := \mathbb{E}_{\tau \sim p_{\pi^*}}[R(\tau) | s_0 = s, a_0 = a]$, the return achieved under the optimal policy π^* when starting in state s and taking action a, with a parameterized Q-function. This requires minimizing the following TD loss:

$$\mathcal{L}_{\text{TD}}(\theta) := \mathbb{E}_{(s,a,r,s') \in \mathcal{D}}[(r + \gamma \max_{a' \in \mathcal{A}} Q_\theta(s', a') - Q_\theta(s, a))^2] \tag{1}$$

Continuous action spaces further require learning a parametric policy $\pi_\phi(a|s)$ that plays the role of the maximizing action in equation 1. This results in a policy objective that must be maximized:

$$\mathcal{J}(\phi) := \mathbb{E}_{s \in \mathcal{D}, a \sim \pi_\phi(\cdot|s)}[Q(s, a)] \tag{2}$$

Here, the dataset of transitions $\mathcal{D}$ evolves as the agent interacts with the environment and both Q_θ and π_ϕ are trained together. These methods make use of function approximation, off-policy learning, and bootstrapping, leading to several instabilities in practice (Sutton, 1988; Van Hasselt et al., 2018).

Offline RL In this setting, we must find a return-maximizing policy from a fixed dataset of transitions collected by an unknown behavior policy μ (Levine et al., 2020). Using TD-learning naively causes the state visitation distribution $d^{\pi_\phi}(s)$ to move away from the distribution of the dataset $d^\mu(s)$. In turn, the policy π_ϕ begins to take actions that are substantially different from those already seen in the data. Offline RL algorithms resolve this distribution-shift by imposing a constraint of the form $D(d^{\pi_\phi} || d^\mu)$, where D is some divergence metric, directly in the TD-learning procedure. The constrained optimization problem now demands additional hyper-parameter tuning and implementation heuristics to achieve any reasonable performance (Kumar et al., 2021). The Decision Diffuser, in comparison, does not have any of these disadvantages. It does not require estimating any kind of Q-function, thereby sidestepping TD methods altogether. It also does not face the risk of distribution-shift because the generative model is trained with maximum-likelihood estimation.

2.2 DIFFUSION PROBABILISTIC MODELS

Diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020) are a specific type of generative model that learn the data distribution $q(\boldsymbol{x})$ from a dataset $\mathcal{D} := \{\boldsymbol{x}^i\}_{0 \leq i < M}$. They have been used most notably for synthesizing high-quality images from text descriptions (Saharia et al., 2022; Nichol et al., 2021). Here, the data-generating procedure is modelled with a predefined forward noising process $q(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t) := \mathcal{N}(\boldsymbol{x}_{t+1}; \sqrt{\alpha_t}\boldsymbol{x}_t, (1 - \alpha_t)\boldsymbol{I})$ and a trainable reverse process $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) := \mathcal{N}(\boldsymbol{x}_{t-1}|\mu_\theta(\boldsymbol{x}_t, t), \Sigma_t)$, where $\mathcal{N}(\mu, \Sigma)$ denotes a Gaussian distribution with mean μ and variance Σ, $\alpha_t \in \mathbb{R}$ determines the variance schedule, $\boldsymbol{x}_0 := \boldsymbol{x}$ is a sample, $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_{N-1}$ are the latents, and $\boldsymbol{x}_N \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ for carefully chosen α_t and long enough T. Starting with Gaussian noise, samples are then iteratively generated through a series of "denoising" steps.

Although a tractable variational lower-bound on $\log p_\theta$ can be optimized to train diffusion models, Ho et al. (2020) propose a simplified surrogate loss:

$$\mathcal{L}_{\text{denoise}}(\theta) := \mathbb{E}_{t \sim [1,N], \boldsymbol{x}_0 \sim q, \epsilon \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})}[||\epsilon - \epsilon_\theta(\boldsymbol{x}_t, t)||^2] \tag{3}$$

The predicted noise $\epsilon_\theta(\boldsymbol{x}_t, t)$, parameterized with a deep neural network, estimates the noise $\epsilon \sim \mathcal{N}(0, I)$ added to the dataset sample $\boldsymbol{x}_0$ to produce noisy $\boldsymbol{x}_t$. This is equivalent to predicting the mean of $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$ since $\mu_\theta(\boldsymbol{x}_t, t)$ can be calculated as a function of $\epsilon_\theta(\boldsymbol{x}_t, t)$ (Ho et al., 2020).

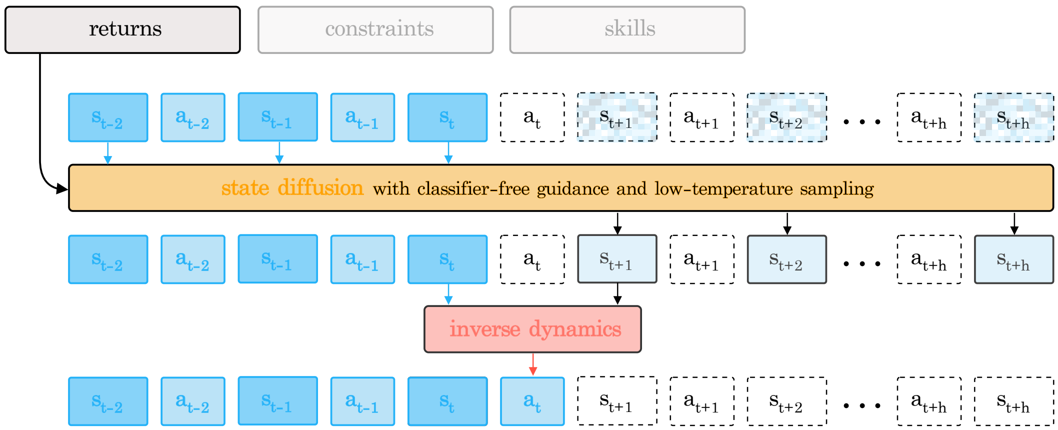

Figure 3: **Planning with Decision Diffuser.** Given the current state s_t and conditioning, Decision Diffuser uses classifier-free guidance with low-temperature sampling to generate a sequence of future states. It then uses inverse dynamics to extract and execute the action a_t that leads to the immediate future state s_{t+1}.

Guided Diffusion Modelling the conditional data distribution $q(\boldsymbol{x}|\boldsymbol{y})$ makes it possible to generate samples with attributes of the label $\boldsymbol{y}$. The equivalence between diffusion models and score-matching (Song et al., 2021), which shows $\epsilon_\theta(\boldsymbol{x}_t, t) \propto \nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t)$, naturally leads to two kinds of methods for conditioning: classifier-guided (Nichol & Dhariwal, 2021) and classifier-free (Ho & Salimans, 2022). The former requires training an additional classifier $p_\phi(\boldsymbol{y}|\boldsymbol{x}_t)$ on noisy data so that samples may be generated at test-time with the perturbed noise $\epsilon_\theta(\boldsymbol{x}_t, t) - s\sqrt{1-\bar{\alpha}_t}\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{y}|\boldsymbol{x}_t)$, where s is referred to as the guidance scale. The latter does not separately train a classifier but modifies the original training setup to learn both a conditional $\epsilon_\theta(\boldsymbol{x}_t, \boldsymbol{y}, t)$ and an unconditional $\epsilon_\theta(\boldsymbol{x}_t, t)$ model for the noise. The unconditional noise is represented, in practice, as the conditional noise $\epsilon_\theta(\boldsymbol{x}_t, \varnothing, t)$ where a dummy value $\varnothing$ takes the place of $\boldsymbol{y}$. The perturbed noise $\epsilon_\theta(\boldsymbol{x}_t, t) + s(\epsilon_\theta(\boldsymbol{x}_t, \boldsymbol{y}, t) - \epsilon_\theta(\boldsymbol{x}_t, t))$ is used to later generate samples.

3 Generative Modeling with the Decision Diffuser

It is useful to solve RL from offline data, both without relying on TD-learning and without risking distribution-shift. To this end, we formulate sequential decision-making as the standard problem of conditional generative modeling:

$$\max_\theta \mathbb{E}_{\tau \sim \mathcal{D}}[\log p_\theta(\boldsymbol{x}_0(\tau)|\boldsymbol{y}(\tau))] \tag{4}$$

Our goal is to estimate the conditional data distribution with p_θ so we can later generate portions of a trajectory $\boldsymbol{x}_0(\tau)$ from information $\boldsymbol{y}(\tau)$ about it. Examples of $\boldsymbol{y}$ could include the return under the trajectory, the constraints satisfied by the trajectory, or the skill demonstrated in the trajectory. We construct our generative model according to the conditional diffusion process:

$$q(\boldsymbol{x}_{t+1}(\tau)|\boldsymbol{x}_t(\tau)), \quad p_\theta(\boldsymbol{x}_{t-1}(\tau)|\boldsymbol{x}_t(\tau), \boldsymbol{y}(\tau)) \tag{5}$$

As usual, q represents the forward noising process while p_θ the reverse denoising process. In the following, we discuss how we may use diffusion for decision making. First, we discuss the modeling choices fof diffusion in Section 3.1. Next, we discuss how we may utilize classifier-free guidance to capture the best aspects of trajectories in Section 3.2. We then discuss the different behaviors that may be implemented with conditional diffusion models in Section 3.3. Finally, we discuss practical training details of our approach in Section 3.4.

3.1 Diffusing over States

In images, the diffusion process is applied across all pixel values in an image. Naïvely, it would therefore be natural to apply a similar process to model the state and actions of a trajectory. However, in the reinforcement learning setting, directly modeling actions using a diffusion process has several practical issues. First, while states are typically continuous in nature in RL, actions are more varied, and are often discrete in nature. Furthermore, sequences over actions, which are often represented as joint torques, tend to be more high-frequency and less smooth, making them much harder to predict and model (Tedrake, 2022). Due to these practical issues, we choose to diffuse only over states, as defined below:

$$\boldsymbol{x}_t(\tau) := (s_k, s_{k+1}, ..., s_{k+H-1})_t \tag{6}$$

Here, t denotes the timestep in the forward process and k denotes the time at which a state was visited in trajectory τ. Moving forward, we will view $\boldsymbol{x}_t(\tau)$ as a noisy sequence of states from a trajectory of length H. We represent $\boldsymbol{x}_t(\tau)$ as a two-dimensional array with one column for each timestep of the sequence.

Acting with Inverse-Dynamics. Sampling states from a diffusion model is not enough for defining a controller. A policy can, however, be inferred from estimating the action a_k that led the state s_k to s_{k+1} for any timestep k in $\boldsymbol{x}_0(\tau)$. Given two consecutive states, we generate an action according to the inverse dynamics model (Agrawal et al., 2016; Pathak et al., 2018):

$$a_k := f_\phi(s_k, s_{k+1}) \tag{7}$$

Note that the same offline data used to train the reverse process p_θ can also be used to learn f_ϕ.

We illustrate in Table 2 how the design choice of directly diffusing state distributions, with an inverse dynamics model to predict action, significantly improves performance over diffusing across both states and actions jointly.

3.2 Planning with Classifier-Free Guidance

Given a diffusion model representing the different trajectories in a dataset, we next discuss how we may utilize the diffusion model for planning. To use the model for planning, it is necessary to additionally condition the diffusion process on characteristics $\boldsymbol{y}(\tau)$. One approach could be to train a classifier $p_\phi(\boldsymbol{y}(\tau)|\boldsymbol{x}_t(\tau))$ to predict $\boldsymbol{y}(\tau)$ from noisy trajectories $\boldsymbol{x}_t(\tau)$. In the case that $\boldsymbol{y}(\tau)$ represents the return under a trajectory, this would require estimating a Q-function in the same way as Janner et al. (2022), which requires a separate, complex dynamic programming procedure.

One approach to avoid dynamic programming is to directly train a conditional diffusion model conditioned on the returns $\boldsymbol{y}(\tau)$ in the offline dataset. However, as our dataset consists of a set of sub-optimal trajectories, the conditional diffusion model will be polluted by such sub-optimal behaviors. To circumvent this issue, we utilize classifier-free guidance (Ho & Salimans, 2022) with low-temperature sampling, to extract high-likelihood trajectories in the dataset. We find that such trajectories correspond to the best set of behaviors in the dataset. Formally, to implement classifier free guidance, a $\boldsymbol{x}_0(\tau)$ is sampled by starting with Gaussian noise $\boldsymbol{x}_T(\tau)$ and refining $\boldsymbol{x}_t(\tau)$ into $\boldsymbol{x}_{t-1}(\tau)$ at each intermediate timestep with the perturbed noise:

$$\hat{\epsilon} := \epsilon_\theta(\boldsymbol{x}_t(\tau), \varnothing, t) + s(\epsilon_\theta(\boldsymbol{x}_t(\tau), \boldsymbol{y}(\tau), t) - \epsilon_\theta(\boldsymbol{x}_t(\tau), \varnothing, t)), \tag{8}$$

where the scalar s applied to $(\epsilon_\theta(\boldsymbol{x}_t(\tau), \boldsymbol{y}(\tau), t) - \epsilon_\theta(\boldsymbol{x}_t(\tau), \varnothing, t))$ seeks to augment and extract the best portions of trajectories in the dataset that exhibit $\boldsymbol{y}(\tau)$.

With these ingredients, sampling from the Decision Diffuser becomes similar to planning in RL. First, we observe a state in the environment. Next, we sample states later into the horizon with our diffusion process conditioned on $\boldsymbol{y}$ and history of last K states observed. Finally, we identify the action that should be taken to reach the most immediate predicted state with our inverse dynamics model. This procedure repeats in a standard receding-horizon control loop described in Algorithm 1 and visualized in Figure 3.

3.3 Conditioning beyond Returns

So far we have not explicitly defined the conditioning variable $\boldsymbol{y}(\tau)$. Though we have mentioned that it can be the return under a trajectory, we may also consider guiding our diffusion process towards sequences of states that satisfy relevant constraints or demonstrate specific behavior. We describe these scenarios in greater detail below.

Maximizing Returns To generate trajectories that maximize return, we condition the noise model on the return of a trajectory so $\epsilon_\theta(\boldsymbol{x}_t(\tau), \boldsymbol{y}(\tau), t) := \epsilon_\theta(\boldsymbol{x}_t(\tau), R(\tau), t)$. These returns are normalized

Algorithm 1 Conditional Planning with the Decision Diffuser

Input: Noise model ϵ_θ, inverse dynamics f_ϕ, guidance scale s, history length K, condition $\boldsymbol{y}$
Initialize $h \leftarrow \texttt{Queue}(\texttt{length} = K)$, $k \leftarrow 0$ // Maintain a history of length K
while not done **do**
  Observe state s; $h.\texttt{insert}(s)$; Initialize $\boldsymbol{x}_T(\tau) \sim \mathcal{N}(0, \alpha I)$
  for $t = T \ldots 1$ **do**
    $\boldsymbol{x}_t(\tau)[:\texttt{length}(h)] \leftarrow h$ // Constrain plan to be consistent with history
    $\hat{\epsilon} \leftarrow \epsilon_\theta(\boldsymbol{x}_t(\tau), t) + s(\epsilon_\theta(\boldsymbol{x}_t(\tau), \boldsymbol{y}, t) - \epsilon_\theta(\boldsymbol{x}_t(\tau), i))$ // Classifier-free guidance
    $(\mu_{t-1}, \Sigma_{t-1}) \leftarrow \texttt{Denoise}(\boldsymbol{x}_t(\tau), \hat{\epsilon})$
    $\boldsymbol{x}_{t-1} \sim \mathcal{N}(\mu_{t-1}, \alpha\Sigma_{t-1})$
  end for
  Extract (s_k, s_{k+1}) from $x_0(\tau)$
  Execute $a_k = f_\phi(s_t, s_{k+1})$; $k \leftarrow k + 1$
end while

to keep $R(\tau) \in [0, 1]$. Sampling a high return trajectory amounts to conditioning on $R(\tau) = 1$. Note that we do not make use of any Q-values, which would then require dynamic programming.

Satisfying Constraints Trajectories may satisfy a variety of constraints, each represented by the set $\mathcal{C}_i$, such as reaching a specific goal, visiting states in a particular order, or avoiding parts of the state space. To generate trajectories satisfying a given constraint $\mathcal{C}_i$, we condition the noise model on a one-hot encoding so that $\epsilon_\theta(\boldsymbol{x}_t(\tau), \boldsymbol{y}(\tau), t) := \epsilon_\theta(\boldsymbol{x}_t(\tau), \mathbb{1}(\tau \in \mathcal{C}_i), t)$. Although we train with an offline dataset in which trajectories satisfy only one of the available constraints, at inference we can satisfy several constraints together.

Composing Skills A skill i can be specified from a set of demonstrations $\mathcal{B}_i$. To generate trajectories that demonstrate a given skill, we condition the noise model on a one-hot encoding so that $\epsilon_\theta(\boldsymbol{x}_t(\tau), \boldsymbol{y}(\tau), t) := \epsilon_\theta(\boldsymbol{x}_t(\tau), \mathbb{1}(\tau \in \mathcal{B}_i), t)$. Although we train with individual skills, we may further compose these skills together during inference.

Assuming we have learned the data distributions $q(\boldsymbol{x}_0(\tau)|\boldsymbol{y}^1(\tau)), \ldots, q(\boldsymbol{x}_0(\tau)|\boldsymbol{y}^n(\tau))$ for n different conditioning variables, we can sample from the composed data distribution $q(\boldsymbol{x}_0(\tau)|\boldsymbol{y}^1(\tau), \ldots, \boldsymbol{y}^n(\tau))$ using the perturbed noise (Liu et al., 2022):

$$\hat{\epsilon} := \epsilon_\theta(\boldsymbol{x}_t(\tau), \emptyset, t) + s\sum_{i=1}^{n}(\epsilon_\theta(\boldsymbol{x}_t(\tau), \boldsymbol{y}^i(\tau), t) - \epsilon_\theta(\boldsymbol{x}_t(\tau), \emptyset, t)) \tag{9}$$

We use this property to compose more than one constraint or skill together at test-time.

3.4 Training the Decision Diffuser

The Decision Diffuser, our conditional generative model for decision-making, is trained in a supervised manner. Given a dataset $\mathcal{D}$ of trajectories, each labeled with the return it achieves, the constraint that it satisfies, or the skill that it demonstrates, we simultaneously train the reverse diffusion process p_θ, parameterized through the noise model ϵ_θ, and the inverse dynamics model f_ϕ with the following loss:

$$\mathcal{L}(\theta, \phi) := \mathbb{E}_{t, \tau \in \mathcal{D}, \beta \sim \text{Bern}(p)}[||\epsilon - \epsilon_\theta(\boldsymbol{x}_t(\tau), (1-\beta)\boldsymbol{y}(\tau) + \beta\emptyset, t)||^2] + \mathbb{E}_{(s,a,s') \in \mathcal{D}}[||a - f_\phi(s, s')||^2]$$

For each trajectory τ, we first sample noise $\epsilon \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and a timestep $t \sim \mathcal{U}\{1, \ldots, T\}$. Then, we construct a noisy array of states $\boldsymbol{x}_t(\tau)$ and finally predict the noise as $\hat{\epsilon}_\theta := \epsilon_\theta(\boldsymbol{x}_t(\tau), \boldsymbol{y}(\tau), t)$. Note that with probability p we ignore the conditioning information and the inverse dynamics is trained with individual transitions rather than trajectories.

Architecture We parameterize ϵ_θ with a temporal U-Net architecture, a neural network consisting of repeated convolutional residual blocks (Janner et al., 2022). This effectively treats a sequence of states $\boldsymbol{x}_t(\tau)$ as an image where the height represents the dimension of a single state and the width denotes the length of the trajectory. We encode the conditioning information $\boldsymbol{y}(\tau)$ as either a scalar or a one-hot vector and project it into a latent variable $z \in \mathbb{R}^h$ with a multi-layer perceptron (MLP). When $\boldsymbol{y}(\tau) = \emptyset$, we zero out the entries of z. We also parameterize the inverse dynamics f_ϕ with an MLP. For implementation details, please refer to the Appendix B.

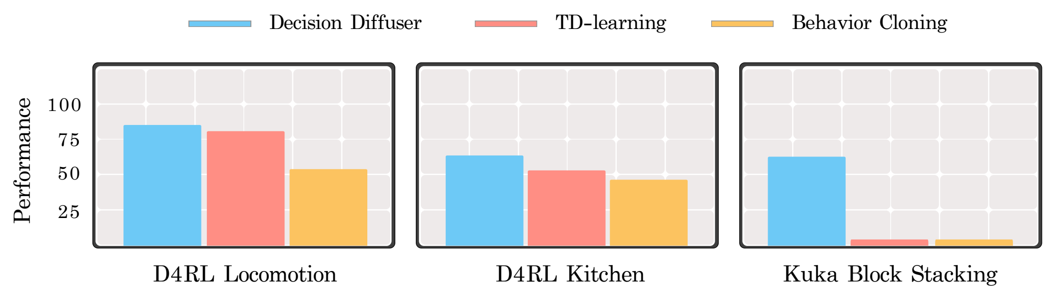

Figure 4: **Results Overview.** Decision Diffuser performs better than both TD learning (CQL) and Behavorial Cloning (BC) across D4RL locomotion tasks, D4RL Kitchen tasks and Kuka Block Stacking tasks (single constraint) using only a conditional generative modeling objective.

Low-temperature Sampling In the denoising step of Algorithm 1, we compute μ_{t-1} and Σ_{t-1} from a noisy sequence of states and a predicted noise. We find that sampling $x_{t-1} \sim \mathcal{N}(\mu_{t-1}, \alpha\Sigma_{t-1})$ where the variance is scaled by $\alpha \in [0, 1)$ leads to better quality sequences (corresponding to sampling lower temperature samples). For a proper ablation study, please refer to Appendix C.

4 Experiments

In our experiments section, we explore the efficacy of the Decision Diffuser on a variety of different decision making tasks (performance illustrated in Figure 4). In particular, we evaluate **(1)** the ability to recover effective RL policies from offline data, **(2)** the ability to generate behavior that satisfies multiple sets of constraints, **(3)** the ability compose multiple different skills together.

4.1 Offline Reinforcement Learning

Setup We first test whether the Decision Diffuser can generate return-maximizing trajectories. To test this, we train a state diffusion process and inverse dynamics model on publicly available D4RL datasets (Fu et al., 2020). We compare with existing offline RL methods, including model-free algorithms like CQL (Kumar et al., 2020) and IQL (Kostrikov et al., 2022), and model-based algorithms such as trajectory transformer (TT, Janner et al. (2021)) and MoReL (Kidambi et al., 2020). We also compare with sequence-models like the Decision Transformer (DT) (Chen et al. (2021) and diffusion models like Diffuser (Janner et al., 2022).

Dataset	Environment	BC	CQL	IQL	DT	TT	MOReL	Diffuser	DD
Med-Expert	HalfCheetah	55.2	91.6	86.7	86.8	**95**	53.3	79.8	90.6 ±1.3
Med-Expert	Hopper	52.5	105.4	91.5	107.6	**110.0**	108.7	107.2	**111.8** ±1.8
Med-Expert	Walker2d	**107.5**	**108.8**	**109.6**	**108.1**	101.9	95.6	**108.4**	**108.8** ±1.7
Medium	HalfCheetah	42.6	44.0	47.4	42.6	46.9	42.1	44.2	**49.1** ±1.0
Medium	Hopper	52.9	58.5	66.3	67.6	61.1	**95.4**	58.5	79.3 ±3.6
Medium	Walker2d	75.3	72.5	78.3	74.0	79	77.8	79.7	**82.5** ±1.4
Med-Replay	HalfCheetah	36.6	**45.5**	**44.2**	36.6	41.9	40.2	42.2	39.3 ±4.1
Med-Replay	Hopper	18.1	95	94.7	82.7	91.5	93.6	96.8	**100** ±0.7
Med-Replay	Walker2d	26.0	77.2	73.9	66.6	**82.6**	49.8	61.2	75 ±4.3
Average		51.9	77.6	77	74.7	78.9	72.9	75.3	**81.8**
Mixed	Kitchen	51.5	52.4	51	-	-	-	-	**65** ±2.8
Partial	Kitchen	38	50.1	46.3	-	-	-	-	**57** ±2.5
Average		44.8	51.2	48.7	-	-	-	-	**61**

Table 1: **Offline Reinforcement Learning Performance.** We show that Decision Diffuser (DD) either matches or outperforms current offline RL approaches on D4RL tasks in terms of normalized average returns (Fu et al., 2020). We report the mean and the standard error over 5 random seeds.

Results Across a broad suite of different offline reinforcement learning tasks, we find that the Decision Diffuser is either competitive or outperforms many of our offline RL baselines (Table 1). It

Hopper-⋆	Diffuser	CondDiffuser	CondMLPDiffuser	Decision Diffuser
Med-Expert	107.6	**111.3**	105.6	**111.8** ± 1.6
Medium	58.5	66.3	54.1	**79.3** ± 3.6
Med-Replay	96.8	76.5	66.5	**100** ± 0.7

Table 2: **Ablations.** Using classifier-free guidance with Diffuser, resulting in `CondDiffuser`, improves performance in 2 (out of 3) environments. Additionally, using inverse dynamics for action prediction in Decision Diffuser improves performance in all 3 environments. `CondMLPDiffuser`, that diffuses over current action given the current state and the target return, doesn't perform as well.

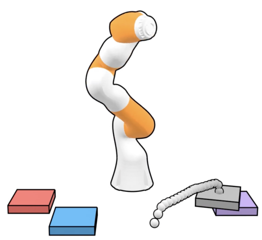

Figure 5: **Kuka Block Stacking task.**

Environment	Diffuser	DD
Single Constraint - Stacking	45.6 ± 3.1	**58.0** ± 3.1
Single Constraint - Rearrangement	58.9 ± 3.4	**62.7** ± 3.1
Single Constraint Average	52.3	**60.4**
Multiple Constraints - Stacking	-	**60.3** ± 3.1
Multiple Constraints - Rearrangement	-	**67.2** ± 3.1
Multiple Constraints Average	-	**63.8**

Table 3: **Block Stacking through Constraint Minimization.** Decision Diffuser (`DD`) improves over Diffuser in generating trajectories satisfying a set of block stacking constraints. It can also flexibly combine multiple constraints during test time.

also outperforms Diffuser and sequence modeling approaches, such as Decision Transformer and Trajectory Transformer. The difference between Decision Diffuser and other methods becomes even more significant on harder D4RL Kitchen tasks which require long-term credit assignment.

To convey the importance of classifier-free guidance, we also compare with the baseline `CondDiffuser`, which diffuses over both state and action sequences as in Diffuser without classifier-guidance. In Table 2, we observe that `CondDiffuser` improves over Diffuser in 2 out of 3 environments. Decision Diffuser further improves over `CondDiffuser`, performing better across all 3 environments. We conclude that learning the inverse dynamics is a good alternative to diffusing over actions. Finally, we compare against `CondMLPDiffuser`, a policy where the current action is denoised according to a diffusion process conditioned on both the state and return. We see that `CondMLPDiffuser` performs the worst amongst diffusion models.

4.2 Constraint Satisfaction

Setup We next evaluate how well we can generate trajectories that satisfy a set of constraints using the Kuka Block Stacking environment (Janner et al., 2022) visualized in Figure 5. In this domain, there are four blocks which can be *stacked* as a single tower or *rearranged* into several towers. A constraint like $\texttt{BlockHeight}(i) > \texttt{BlockHeight}(j)$ requires that block i be placed above block j. We train the Decision Diffuser from $10,000$ expert demonstrations each satisfying one of these constraints. We randomize the positions of these blocks and consider two tasks at inference: sampling trajectories that satisfy a single constraint seen before in the dataset or satisfy a group of constraints for which demonstrations were never provided. In the latter, we ask the Decision Diffuser to generate trajectories so $\texttt{BlockHeight}(i) > \texttt{BlockHeight}(j) > \texttt{BlockHeight}(k)$ for three of the four blocks i, j, k. For more details, please refer to Appendix D.

Results In both the stacking and rearrangement settings, Decision Diffuser satisfies single constraints with greater success rate than Diffuser (Table 3). We also compare with `BCQ` (Fujimoto et al., 2019) and `CQL` (Kumar et al., 2020), but they consistently fail to stack or rearrange the blocks and get a success rate of 0.0. Unlike these baselines, our method can just as effectively satisfy several constraints together according to Equation 9. For a visualization of these generated trajectories, please see the website https://sites.google.com/view/decisiondiffuser/.

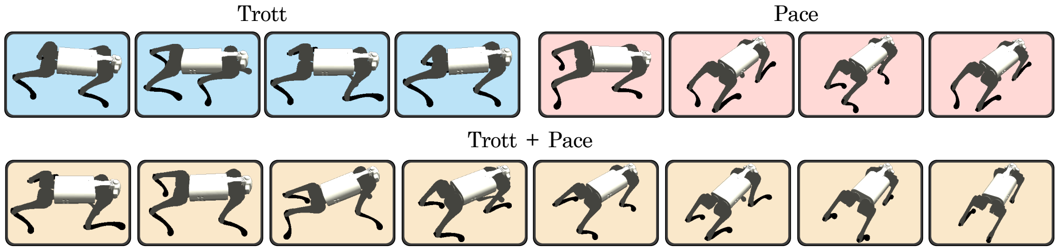

Figure 6: **Composing Movement Skills.** Decision Diffuser is not only able to imitate individual running gaits using expert demonstrations but also compose multiple different skills together during test time. The results are best illustrated by videos viewable at https://sites.google.com/view/decisiondiffuser/.

4.3 Skill Composition

Setup Finally, we explore how will we may compose different skills together. We consider the Unitree-go-running environment (Margolis & Agrawal, 2022), where a quadruped robot can be found running with various gaits, like bounding, pacing, and trotting. We explore if it is possible to generate trajectories that transition between these gaits after only training on individual gaits separately. For each gait, we collect a dataset of 2500 demonstrations. Then, we train the Decision Diffuser to imitate each of these these trajectories.

Results During testing, we compose the noise model of our reverse diffusion process according to equation 9. This allows us to sample trajectories of the quadruped robot with entirely new running behavior. Figure 6 shows a trajectory that begins with trotting but ends with pacing. Please see the Appendix E for more visualizations of composing running gaits together.

5 Related Work

Diffusion Models Diffusion Models have shown great promise in learning generative models of image and text data (Saharia et al., 2022; Nichol et al., 2021; Nichol & Dhariwal, 2021). It formulates the data sampling process as an iterative denoising procedure (Sohl-Dickstein et al., 2015; Ho et al., 2020). The denoising procedure can be alternatively interpreted as parameterizing the gradients of the data distribution (Song et al., 2021) optimizing the score matching objective (Hyvärinen, 2005) and Energy-Based Models (Du & Mordatch, 2019; Nijkamp et al., 2019; Grathwohl et al., 2020). To generate data samples (eg: images) conditioned on some additional information (eg:text), prior works (Nichol & Dhariwal, 2021) have learned a classifier to facilitate the conditional sampling. More recent works (Ho & Salimans, 2022) have argued to leverage gradients of an implicit classifier, formed by the difference in score functions of a conditional and an unconditional model, to facilitate conditional sampling. The resulting classifier-free guidance has been shown to generate better conditional samples than classifier-based guidance. All these above mentioned works have mostly focused on generation of text or images. Janner et al. (2022) used diffusion models to learn a generative model of trajectories (consisting of states and actions) for decision-making. However, Janner et al. (2022) relies on learning a Q-function to guide the trajectory model towards high-return trajectories, thereby still retaining the instabilities and the complexities of offline RL pipeline. In contrast, Decision Diffuser learns a conditional diffusion model over state sequences and uses classifier-free guidance for sampling high-return state sequences, thereby side-stepping learning of Q-function and avoiding any complexities of offline RL.

Reward Conditioned Policies Prior works (Kumar et al., 2019; Schmidhuber, 2019; Srivastava et al., 2019; Emmons et al., 2021; Chen et al., 2021) have studied learning of reward conditioned policies via reward conditioned behavioral cloning. Chen et al. (2021) used a transformer (Vaswani et al., 2017) to model the reward conditioned policies and obtained a performance competitive with offline RL approaches. Emmons et al. (2021) obtained similar performance as Chen et al. (2021) without using a transformer policy but relied on careful capacity tuning of MLP policy. In contrast to these works, in addition to modeling returns, Decision Diffuser can also model constraints or skills and generate novel behaviors by flexibly combining multiple constraints or skills during test time.

6 Discussion

We propose Decision Diffuser, a conditional generative model for sequential decision making. It frames offline sequential decision making as conditional generative modeling and sidesteps the need of reinforcement learning, thereby making the decision making pipeline simpler. By sampling for high returns, it is able to capture the best behaviors in the dataset and outperforms existing offline RL approaches on standard benchmarks (such as D4RL). In addition to returns, it can also be conditioned on constraints or skills and can generate novel behaviors by flexibly combining constraints or composing skills during test time.

In this work, we focused on offline sequential decision making, thus circumventing the need for exploration. Using ideas from Zheng et al. (2022), future works could look into online fine-tuning of Decision Diffuser by leveraging entropy of the state-sequence model for exploration. While our work focused on state based environments, it can be extended to image based environments by performing the diffusion in latent space, rather than observation space, as done in Rombach et al. (2022).

References

Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *Advances in neural information processing systems*, 29, 2016.

Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. Opal: Offline primitive discovery for accelerating offline reinforcement learning. *arXiv preprint arXiv:2010.13611*, 2020.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems*, 2021.

Robert Dadashi, Shideh Rezaeifar, Nino Vieillard, Léonard Hussenot, Olivier Pietquin, and Matthieu Geist. Offline reinforcement learning with pseudometric learning. *arXiv preprint arXiv:2103.01948*, 2021.

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

Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021.

Scott Emmons, Benjamin Eysenbach, Ilya Kostrikov, and Sergey Levine. Rvs: What is essential for offline rl via supervised learning? *arXiv preprint arXiv:2112.10751*, 2021.

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.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, 2019.

Dibya Ghosh, Anurag Ajay, Pulkit Agrawal, and Sergey Levine. Offline rl policies should be trained to be adaptive. In *International Conference on Machine Learning*, pp. 7513–7530. PMLR, 2022.

Will Grathwohl, Kuan-Chieh Wang, Joern-Henrik Jacobsen, David Duvenaud, and Richard Zemel. Learning the stein discrepancy for training and evaluating energy-based models without sampling. In *International Conference on Machine Learning*, 2020.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, 2020.

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

Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pp. 991–1002. PMLR, 2022.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, 2021.

Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, 2022.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. MOReL: Model-based offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

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

Ilya Kostrikov, Jonathan Tompson, Rob Fergus, and Ofir Nachum. Offline reinforcement learning with fisher divergence critic regularization. *arXiv preprint arXiv:2103.08050*, 2021.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit Q-learning. In *International Conference on Learning Representations*, 2022.

Aviral Kumar, Xue Bin Peng, and Sergey Levine. Reward-conditioned policies. *arXiv preprint arXiv:1912.13465*, 2019.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

Aviral Kumar, Anikait Singh, Stephen Tian, Chelsea Finn, and Sergey Levine. A workflow for offline model-free robotic reinforcement learning. *arXiv preprint arXiv:2109.10813*, 2021.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *arXiv preprint arXiv:2206.14858*, 2022.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Nan Liu, Shuang Li, Yilun Du, Antonio Torralba, and Joshua B Tenenbaum. Compositional visual generation with composable diffusion models. *arXiv preprint arXiv:2206.01714*, 2022.

Gabriel Margolis and Pulkit Agrawal. Walk these ways: Gait-conditioned policies yield diversified quadrupedal agility. In *Conference on Robot Learning*, 2022.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, et al. Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746*, 2020.

Diganta Misra. Mish: A self regularized non-monotonic neural activation function. In *British Machine Vision Conference*, 2019.

Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, 2021.

Erik Nijkamp, Mitch Hill, Song-Chun Zhu, and Ying Nian Wu. Learning non-convergent non-persistent short-run MCMC toward energy-based model. In *Advances in Neural Information Processing Systems*, 2019.

Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A Efros, and Trevor Darrell. Zero-shot visual imitation. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 2050–2053, 2018.

Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley series in probability and statistics. John Wiley Sons Inc., Hoboken, NJ, 1995.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 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.

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.

Juergen Schmidhuber. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, 2015.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021.

Rupesh Kumar Srivastava, Pranav Shyam, Filipe Mutz, Wojciech Jaśkowski, and Jürgen Schmidhuber. Training agents using upside-down reinforcement learning. *arXiv preprint arXiv:1912.02877*, 2019.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 1988.

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

Russ Tedrake. *Underactuated Robotics*. 2022. URL http://underactuated.mit.edu.

Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Homer Walke, Jonathan Yang, Albert Yu, Aviral Kumar, Jedrzej Orbik, Avi Singh, and Sergey Levine. Don't start from scratch: Leveraging prior data to automate robotic reinforcement learning. *arXiv preprint arXiv:2207.04703*, 2022.

Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

Yuxin Wu and Kaiming He. Group normalization. In *European Conference on Computer Vision*, 2018.

Adam Yala, Peter G Mikhael, Constance Lehman, Gigin Lin, Fredrik Strand, Yung-Liang Wan, Kevin Hughes, Siddharth Satuluru, Thomas Kim, Imon Banerjee, et al. Optimizing risk-based breast cancer screening policies with reinforcement learning. *Nature Medicine*, 28(1):136–143, 2022.

Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. *arXiv preprint arXiv:2202.05607*, 2022.

Appendix

In this appendix, we discuss details of the illustrative example in Section A. Next, we discuss hyperparameters and architectural details in Section B. We analyze the importance of low temperature sampling in Section C. Finally, we provide details of the Kuka Block Stacking environment in Section D and the Unitree environment in Section E.

A Illustrative Example

Setup In linear system robot navigation, Decision Diffuser is trained on 1000 expert trajectories either satisfying the constraint $\|s_T\| \leq R$ ($R = 1$) or the constraint $\|s_T\| \geq r$ ($r = 0.7$). Here, $s_T = [x_T, y_T]$ represents the final robot state in a trajectory, specifying its final 2d position. The maximum trajectory length is 50. During test time, Decision Diffuser is asked to generate trajectories satisfying $\|s_T\| \leq R$ and $\|s_T\| \geq r$ to test its ability to satisfy single constraints. Furthermore, Decision Diffuser is also asked to generate trajectories satisfying $r \leq \|s_T\| \leq R$ to test its ability to satisfy combined constraints.

Results Figure 2 shows that Decision Diffuser learns to generate trajectories perfectly (i.e. with 100% success rate) satisfying single constraints in linear system robot navigation. Furthermore, it learns to generate trajectories satisfying the composed constraint in linear system robot navigation with $91.3\%(\pm 2.6\%)$ accuracy where the standard error is calculated over 5 random seeds.

B Hyperparameter and Architectural details

In this section, we describe various architectural and hyperparameter details:

- We represent the noise model ϵ_θ with a temporal U-Net (Janner et al., 2022), consisting of a U-Net structure with 6 repeated residual blocks. Each block consisted of two temporal convolutions, each followed by group norm (Wu & He, 2018), and a final Mish nonlinearity (Misra, 2019). Timestep and condition embeddings, both 128-dimensional vectors, are produced by separate 2-layered MLP (with 256 hidden units and Mish nonlinearity) and are concatenated together before getting added to the activations of the first temporal convolution within each block. We borrow the code for temporal U-Net from https://github.com/jannerm/diffuser.
- We represent the inverse dynamics f_ϕ with a 2-layered MLP with 512 hidden units and ReLU activations.
- We train ϵ_θ and f_ϕ using the Adam optimizer (Kingma & Ba, 2015) with a learning rate of $2e-4$ and batch size of 32 for $1e6$ train steps.
- We choose the probability p of removing the conditioning information to be 0.25.
- We use $T = 100$ diffusion steps.
- We use a planning horizon H of 100 in all the D4RL locomotion tasks, 56 in D4RL kitchen tasks, 128 in Kuka block stacking, 56 in unitree-go-running tasks and 50 in the illustrative example.
- We use a guidance scale $s \in \{1.2, 1.4, 1.6, 1.8\}$ but the exact choice varies by task.
- We choose $\alpha = 0.5$ for low temperature sampling.
- We choose context length $K = 20$.

C Importance of low temperature sampling

In Algorithm 1, we compute μ_{t-1} and Σ_{t-1} from a noisy sequence of states and predicted noise. We find that sampling $x_{t-1} \sim \mathcal{N}(\mu_{t-1}, \alpha\Sigma_{t-1})$ (where $\alpha \in [0, 1)$) with a reduced variance produces high-likelihood state sequences. We refer to this as low-temperature sampling. To empirically show its importance, we compare performances of Decision Diffuser with different values of α (Table A1). We show that low temperature sampling ($\alpha = 0.5$) gives the best average returns. However, reducing the α to 0 eliminates the entropy in sampling and leads to lower returns. On the other hand, $\alpha = 1.0$ leads to a higher variance in terms of returns of the trajectories.

Decision Diffuser	Hopper-Medium-Expert
$\alpha = 0$	104.3 ± 0.7
$\alpha = 0.5$	**111.8** ± 1.6
$\alpha = 1.0$	107.1 ± 3.5

Table A1: Low-temperature sampling ($\alpha = 0.5$) allows us to get high return trajectories consistently. While $\alpha = 1.0$ leads to a higher variance in returns of the trajectories, $\alpha = 0.0$ eliminates entropy in the sampling and leads to lower returns.

D Kuka Block Stacking

In the Kuka blocking stacking environment, the underlying goal is to stack a set of blocks on top of each other. Models have trained on a set of demonstration data, where a set of 4 blocks are sequentially stacked on top of each other to form a block tower.

We construct state-space plans of length 128. Following (Janner et al., 2022), we utilize a close-loop controller to generate actions for each state in our state-space plan (controlling the 7 degrees of freedom in joints). The total maximum trajectory length plan in Kuka block stacking is 384. We detail differences between the two consider conditional stacking environments below:

- **Stacking** In the stacking environment, at test time we wish to again construct a tower of four blocks.
- **Rearrangement** In the rearrangement environment, at test time wish to stack blocks in a configuration where a set of blocks are above a second set. This set of stack-place relations may not precisely correspond to a single block tower (can instead construct two block towers), making this environment an out-of-distribution challenge.

In addition to Diffuser (Janner et al., 2022), we used goal-conditioned variants of `CQL` (Kumar et al., 2020) and `BCQ` (Fujimoto et al., 2019) as baselines for the block stacking and rearrangement with single constraint. However, they get a success rate of 0.0.

E Unitree Go Running

We consider Unitree-go-running environment (Margolis & Agrawal, 2022) where a quadruped robot runs in 3 different gaits: bounding, pacing, and trotting. The state space is 56 dimensional, the action space is 12 dimensional, and the maximum trajectory length is 250.

As described in Section 4.3, we train Decision Diffuser on expert trajectories demonstrating individual gaits. Figure A1 shows the ability of Decision Diffuser to imitate bounding, trotting and pacing and their combinations.

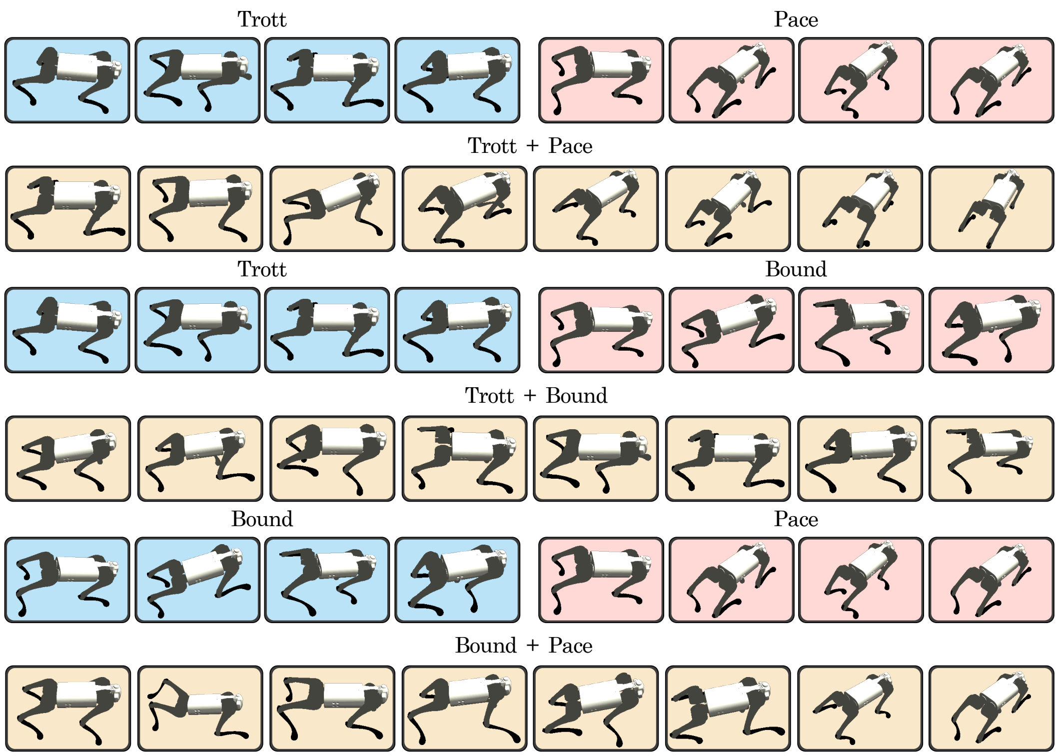

Figure A1: **Composing Movement Skills.** Decision Diffuser is not only able to imitate individual running gaits using expert demonstrations but also compose multiple different skills together during test time. The results are best illustrated by videos viewable at https://sites.google.com/view/decisiondiffuser/.