BAYESIAN FLOW NETWORKS FOR OFFLINE TRAJECTORY PLANNING

PREPRINT

Ludvig Killingberg
Norwegian University of Science and Technology
ludvig.killingberg@ntnu.no

Helge Langseth
Norwegian University of Science and Technology

ABSTRACT

Offline reinforcement learning (RL) leverages static datasets to learn decision policies without real-time environment interaction. While recent sequence-modeling approaches rely on continuous diffusion models for trajectory synthesis, applying these methods to discrete planning tasks requires a categorical formulation rather than the standard Gaussian construction. We present BFN-RL, a unified generative modeling framework for offline RL based on Bayesian Flow Networks (BFNs). By iteratively evolving distribution parameters rather than noisy data instances, BFN-RL natively models both discrete and continuous trajectory spaces within a single probabilistic formulation. The categorical planner generates future state sequences, and a learned inverse-dynamics model converts consecutive generated states into actions. Evaluations in discrete planning and continuous control show that BFN-RL can generate effective trajectories across both categorical and continuous state spaces. Our results establish BFNs as a versatile generative foundation for offline trajectory planning across data modalities.

1 Introduction

Offline reinforcement learning (RL) [Sutton and Barto, 2018, Levine et al., 2020] is a powerful paradigm that leverages static, previously collected datasets to learn effective decision policies without requiring real-time environment interaction. By eliminating the safety hazards associated with online exploration, offline RL is particularly suited for high-risk domains such as autonomous driving and medical decision-making. In recent years, framing offline RL as a conditional sequence-modeling task [Janner et al., 2021, Chen et al., 2021] has emerged as a compelling alternative to traditional value-based methods, which frequently suffer from value overestimation on out-of-distribution state-action pairs [Agarwal et al., 2020, Levine et al., 2020]. By viewing trajectory generation through the lens of conditional generative modeling, sequence-based agents synthesize high-return trajectories by capturing complex temporal dependencies across long horizons [Janner et al., 2022, Ajay et al., 2023].

Despite their empirical success, contemporary sequence-modeling approaches rely almost exclusively on Gaussian Denoising Diffusion Probabilistic Models (DDPMs) [Ho et al., 2020, Janner et al., 2022, Ajay et al., 2023]. While diffusion models excel in continuous control domains, discrete states and actions require a categorical formulation rather than the standard Gaussian construction [Austin et al., 2021, Lou et al., 2023]. This motivates studying a generative framework that handles both data types natively.

To address this limitation, we present *BFN-RL*, an offline reinforcement learning framework grounded in Bayesian Flow Networks (BFNs) [Graves et al., 2023]. Unlike diffusion models that iteratively denoise corrupted data instances, BFNs operate by updating the parameters of an input distribution via Bayesian inference driven by continuous-time parameter flows [Graves et al., 2023]. This formulation yields a unified generative paradigm that natively handles categorical, continuous, and discretized variables within the same probabilistic framework.

Our main contributions are summarized as follows:

- **Unified Generative Planning Framework:** We introduce BFN-RL, establishing a parameter-flow paradigm for offline sequence modeling that operates seamlessly across both discrete and continuous state-action spaces [Graves et al., 2023].
- **Categorical Trajectory Planning:** We combine a categorical BFN state-sequence model with learned inverse dynamics to obtain a discrete planner.
- **Cross-Domain Evaluation:** We demonstrate that the same BFN planning formulation is viable in both discrete and continuous control problems.

2 Preliminaries

2.1 Reinforcement Learning

Reinforcement learning (RL) is a framework for learning to make decisions in an environment [Sutton and Barto, 2018]. The interactions with the environment are modeled as a Markov decision process (MDP), which is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the transition function, $\mathcal{R}$ is the reward function, and γ is the discount factor. In an environment where the agent performs the action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$, the next state, $s' \in \mathcal{S}$ is sampled from $\mathcal{P}(s, a)$, i.e., is only dependent on the current state and action, not the history of previous states and actions. In other words, the domain adheres to the Markov property. Let r_t denote the reward received at time t, and let $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ be the discounted cumulative reward obtained starting from time t. Now, one goal of RL is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected return $\mathbb{E}[R_t]$, where the expectation is taken over the uncertainty defined by both the transition function and the stochastic strategy π.

The exploration-exploitation trade-off is a fundamental challenge in reinforcement learning, typically associated with online learning scenarios where agents iteratively interact with an environment to learn optimal policies. Exploration involves sampling actions to gather information about the environment, potentially leading to the discovery of better strategies, while exploitation entails leveraging known information in an attempt to maximize the expected returns. Much of the research in online RL is dedicated to striking a balance between exploration and exploitation, devising algorithms that effectively navigate this trade-off to converge to optimal or near-optimal policies.

2.2 Offline RL

In the realm of offline reinforcement learning, the primary objective is to learn effective policies from a fixed dataset, without the need for online interactions [Levine et al., 2020]. In this context, where the exploration aspect is inherently absent, the focus shifts towards effectively utilizing the available dataset to optimize policies. Traditionally, RL has been concerned with estimating stationary policies or single-step models, leveraging the Markov property to factorize problems over time. However, applying standard RL methods to offline settings is challenging because methods relying on an estimated value function often suffer from overestimating the value of out-of-distribution state-action pairs. Various methods have been proposed to address this issue, including constraining the policy to be close to the data distribution [Peters et al., 2010] or using a conservative value function [Kumar et al., 2020]. Our solution, on the other hand, is to produce a sequence of steps that will be generated conditionally on the objective of the RL agent. An intriguing perspective emerges when we view RL through the lens of sequence modeling. Instead of treating it as a specialized domain, we can consider RL as a generic sequence modeling problem. The crux of this viewpoint lies in producing a sequence of actions that leads to a sequence of high rewards. Earlier work has solved this by conditioning the model on returns such that trajectories with high returns can be generated in online settings [Ajay et al., 2023, Janner et al., 2021]. By adopting this perspective, we can simplify design decisions and dispense with many components commonly found in offline RL algorithms. This approach not only demonstrates flexibility across various tasks such as long-horizon dynamics prediction, imitation learning, goal-conditioned RL, and offline RL but also yields state-of-the-art planners in sparse-reward, long-horizon scenarios [Janner et al., 2022, Ajay et al., 2023].

2.3 Denoising Diffusion Probabilistic Models

Since the current state of the art in this domain [Janner et al., 2022, Ajay et al., 2023] rely on diffusion models as the generative model, we will give a brief introduction to denoising diffusion probabilistic models (DDPMs) [Ho et al., 2020]. This will also serve as a backdrop for our discussion of Bayesian flow networks, which follows in Section 2.5. DDPMs are a type of generative model inspired by non-equilibrium thermodynamics. The model is defined by a *forward process* that slowly adds Gaussian noise to data, and its *reverse*, that amounts to learning to iteratively denoise the noisy data. Diffusion models have primarily been used for image generation, but have also shown state-of-the-art performance in other domains, like video generation and 3D model [Ho et al., 2022, Luo and Hu, 2021].

Given data $\mathbf{x}_0 \sim q(\mathbf{x})$, we define the *forward process* to produce a sequence of noisy samples $\mathbf{x}_1, \ldots, \mathbf{x}_K$,

$$q(\mathbf{x}_k|\mathbf{x}_{k-1}) = \mathcal{N}(\mathbf{x}_k; \sqrt{1-\beta_k}\mathbf{x}_{k-1}, \beta_k \boldsymbol{I}).$$

where $\{\beta_i \in (0,1)\}_{i=1}^K$ is a carefully chosen variance schedule. A nice property of the forward process is that we can directly sample $\mathbf{x}_k$ at any step i, because the distribution $q(\mathbf{x}_k|\mathbf{x}_0)$ can be derived using the property that a sum of uncorrelated normally distributed random variables is normally distributed. Let $a_k = 1 - \beta_k$ and $\overline{a}_k = \prod_{i=1}^k a_k$, then

$$q(\mathbf{x}_k|\mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_k; \sqrt{\overline{a}_k}\,\mathbf{x}_0, (1-\overline{a}_k)\,\boldsymbol{I}\right).$$

Note also that

$$q(\mathbf{x}_{k-1}|\mathbf{x}_k, \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_{k-1}; \tilde{\boldsymbol{\mu}}(\mathbf{x}_k, \mathbf{x}_0), \tilde{\beta}_k \boldsymbol{I}\right), \tag{1}$$

where

$$\tilde{\boldsymbol{\mu}}(\mathbf{x}_k, \mathbf{x}_0) = \frac{\sqrt{\overline{a}_{k-1}}\beta_k}{1-\overline{a}_k}\mathbf{x}_0 + \frac{\sqrt{a_k}(1-\overline{a}_{k-1})}{1-\overline{a}_k}\mathbf{x}_k, \qquad \tilde{\beta}_k = \beta_k \frac{1-\overline{a}_{k-1}}{1-\overline{a}_k}. \tag{2}$$

While the forward process creates a noisy representation of data, the *reverse process* aims to iteratively recreate samples from noise by modeling and then sampling from $q(\mathbf{x}_{k-1}|\mathbf{x}_k)$. Let $p_\theta(\mathbf{x}_{k-1}|\mathbf{x}_k)$ be a parameterized approximation of $q(\mathbf{x}_{k-1}|\mathbf{x}_k)$. This means that we define a neural network model with trainable parameters $\boldsymbol{\theta}$ that outputs $\boldsymbol{\mu}_\theta(\mathbf{x}_k, k)$ and $\Sigma_\theta(\mathbf{x}_k, k)$ so that

$$p_\theta(\mathbf{x}_{k-1}|\mathbf{x}_k) = \mathcal{N}\left(\mathbf{x}_{k-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_k, k), \Sigma_\theta(\mathbf{x}_k, k)\right).$$

Ho et al. [2020] chose to fix the variance term $\boldsymbol{\Sigma}_\theta(\mathbf{x}_k, k)$ as a constant $\sigma_k^2 = \tilde{\beta}_k$, see Eq. (2). We therefore only look at how $\boldsymbol{\mu}_\theta(\mathbf{x}_k, k)$ is estimated. First, we consider the identity

$$\tilde{\boldsymbol{\mu}}_k(\mathbf{x}_k, \mathbf{x}_0) = \frac{1}{\sqrt{a_k}}\left(\mathbf{x}_k - \frac{1-a_k}{\sqrt{1-\overline{a}_k}}\boldsymbol{\epsilon}_k\right),$$

where $\boldsymbol{\epsilon}_k \sim \mathcal{N}(0, \boldsymbol{I})$; cf. Eqs. (1) and (2). Since $\mathbf{x}_k$ is known during training, we can choose to predict $\boldsymbol{\epsilon}_k$, rather than $\tilde{\boldsymbol{\mu}}_k$ directly. Empirically, this has shown better results. Let us define $\boldsymbol{\epsilon}_\theta(\mathbf{x}, k)$ as a model that predicts the noise, $\boldsymbol{\epsilon}_k$. This means that we can define $\boldsymbol{\mu}_\theta(\mathbf{x}_k, k) = \frac{1}{\sqrt{a_k}}\left(\mathbf{x}_k - \frac{1-a_k}{\sqrt{1-\overline{a}_k}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_k, k)\right)$. Ho et al. [2020] derive the following loss function to minimize the difference between $\boldsymbol{\mu}_\theta$ and $\tilde{\boldsymbol{\mu}}$:

$$L(\theta) = \mathbb{E}_{k\sim[1,K],\mathbf{x}_0,\boldsymbol{\epsilon}_k}\left[\frac{\beta_k^2}{2\sigma_k^2 a_k(1-\overline{a}_k)}\left\|\boldsymbol{\epsilon}_k - \boldsymbol{\epsilon}_\theta(\mathbf{x}_k, k)\right\|^2\right].$$

They also present the following simplified loss function that turns out to give better empirical results:

$$L(\theta) = \mathbb{E}_{k\sim[1,K],\mathbf{x}_0,\epsilon_k}\left\|\epsilon_k - \epsilon_\theta(\mathbf{x}_k, k)\right\|^2.$$

2.4 Guided Diffusion

We will discuss three ways diffusion models can condition on variables. The first, *classifier-guided* diffusion [Dhariwal and Nichol, 2021], takes as its starting point that we have a trained probabilistic classifier that classifies objects $\mathbf{x}_k$ during the reverse process. We want to use this to produce an object $\mathbf{x}_0$ that is classified as belonging to a predefined class y. The approach uses the gradients of this classifier's allocated log-likelihood to the class y wrt. $\mathbf{x}_k$ to "push" the reverse process towards objects that are aligned with the conditioning information. This method has the advantage that the diffusion model does not have to be trained with conditioning variables, the guidance is only related to the reverse and only needs the classifier to be trained on conditioning information. A model predictor $\overline{\boldsymbol{\epsilon}}_\theta$, guided by a classifier $h(y|\mathbf{x}_k, k)$ meant to estimate the probability that the noisy datapoint $\mathbf{x}_k$ belongs to class y, would assume the following form:

$$\overline{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_k, k, y) = \boldsymbol{\epsilon}_\theta(\mathbf{x}_k, k) - w\sigma_k \nabla_{\mathbf{x}_k} \log h(y|\mathbf{x}_k, k),$$

where w is a hyper-parameter controlling the strength of the guidance.

Secondly, *classifier-free guidance* [Ho and Salimans, 2021], plugs the conditioning variable directly into the denoising network as an auxiliary input variable during training. At test time, the auxiliary variable can be set to the conditioning value. In this setting, the model predictor takes the following form:

$$\tilde{\boldsymbol{\epsilon}}(\mathbf{x}_k, k, y) = (w+1)\boldsymbol{\epsilon}_\theta(\mathbf{x}_k, k, y) - w\boldsymbol{\epsilon}_\theta(\mathbf{x}_k, k),$$

where we again use w to denote the hyper-parameter that controls the strength of the guidance. Classifier-free guidance has shown better practical performance than classifier-guided diffusion [Ho and Salimans, 2021].

Finally, we can also employ inpainting [Lugmayr et al., 2022] to condition on partial observations. In the context of image generation, this implies conditioning on some pixels within the image. Consider an image $\mathbf{x}_0$ divided into known pixels $\mathbf{x}_0^{\text{known}}$ and unknown pixels $\mathbf{x}_0^{\text{unknown}}$, and a mask $\boldsymbol{m}$ defining which pixels are known. During the reverse process, we define

$$\mathbf{x}_{k-1}^{\text{known}} = \sqrt{\overline{a}_k}\mathbf{x}_0 + \sqrt{1-\overline{a}_k}\boldsymbol{z}, \qquad \boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

i.e., $\mathbf{x}_{k-1}^{\text{known}}$ is chosen equal to what the forward process would have produced had it started from the known parts of the image. The unknown pixels at step k, $\mathbf{x}_{k-1}^{\text{unknown}}$, are computed in standard fashion:

$$\mathbf{x}_{k-1}^{\text{unknown}} = \frac{1}{\sqrt{a_k}}\left(\mathbf{x}_k - \frac{\beta_k}{\sqrt{1-\overline{a}_k}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_k, k)\right) + \sigma_k \boldsymbol{z}, \qquad \boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

Finally, we have:

$$\mathbf{x}_{k-1} = \boldsymbol{m} \odot \mathbf{x}_{k-1}^{\text{known}} + (1-\boldsymbol{m}) \odot \mathbf{x}_{k-1}^{\text{unknown}}, \tag{3}$$

where $\odot$ is elementwise multiplication.

2.5 Bayesian Flow Networks

While there are variations of diffusion models that model discrete data [Lou et al., 2023, Austin et al., 2021], these models are not considered state of the art when it comes to generating high-quality discrete data [Graves et al., 2023]. In an attempt to remedy this shortcoming, Graves et al. [2023] introduced *Bayesian flow networks* (BFNs), a novel generative model capable of generating continuous, discrete, and discretized data. BFNs resemble diffusion models in that they generate data in an iterative process. Unlike diffusion models, however, the BFN analogy to the diffusion models' reverse process iteratively evolves *distribution parameters*, not noised versions of data. The high-level idea is to start from a prior distribution and iteratively update the distribution conditioned on a data point sampled from a noisy version of the previous distribution. BFNs have been shown to perform well on discrete data [Graves et al., 2023], and are therefore a more natural choice than diffusion models for planning in discrete state spaces.

Figure 1 illustrates the idea of Bayesian flow networks. At each step i, the parameters of a distribution ($\boldsymbol{\theta}_i$) are updated with noisy samples from the data, $\boldsymbol{y}_i$. The level of added noise decreases for each step and is at step i dictated by the accuracy α_i. The parameters $\boldsymbol{\theta}_i$ are defined as the Bayesian update of the parameters at the previous step, $\boldsymbol{\theta}_{i-1}$, with observation noise parameterized by α_i using a predetermined update rule $h(\cdot)$. While each evolved distribution is distinct from the data distribution, the idea is that the compound of all generated distributions should approximate the data distribution. To sample from the data distribution, an initial uninformative distribution is iteratively updated, gradually concentrating around a single data point. Once the distribution has evolved sufficiently, a single sample from this nearly degenerate distribution will closely approximate a sample from the data distribution.

$$\boldsymbol{\theta}_0 \longrightarrow \cdots \longrightarrow \boldsymbol{\theta}_i \overset{\text{def}}{=} h(\boldsymbol{\theta}_{i-1}, \boldsymbol{y}_{i-1}, \alpha_{i-1}) \longrightarrow \boldsymbol{\theta}_{i+1} \overset{\text{def}}{=} h(\boldsymbol{\theta}_i, \boldsymbol{y}_i, \alpha_i) \longrightarrow \cdots \longrightarrow \boldsymbol{\theta}_N$$

Figure 1: Generative process for Bayesian flow networks.

Figure 2 represents the training process of Bayesian Flow Networks. The aim is to iteratively update the parameters of a distribution, $\boldsymbol{\theta}$, beginning with a prior distribution $\boldsymbol{\theta}_0$, so that eventually, sampling once from this distribution mirrors sampling from the data it is trained on. In this process, accuracy α_i refers to how well the updated parameters reflect the true data after observing information. The accuracy quantifies the expected quality of each update, meaning how much closer, in expectation, the updated distribution is to the true data distribution.

This training process involves the following key steps:

1. **Generate $\boldsymbol{\theta}_i$:** Given a datapoint $\boldsymbol{x}$, the parameters $\boldsymbol{\theta}_0$ can be updated i times with noisy data to create parameters $\boldsymbol{\theta}_i$. We will later see that accuracies are additive and that $\boldsymbol{\theta}_i$ can be generated in a single step with accuracy $\sum_{j=0}^{i-1} \alpha_j$.
2. **Neural network transformation:** The parameters $\boldsymbol{\theta}_i$ are passed through a neural network, with weights $\boldsymbol{\omega}$, which outputs the parameters of a new distribution. This is referred to as the *output distribution* p_O.
3. **Addition of noise:** A new *sender distribution* p_S, is created by adding noise to the data according to a predefined schedule. Meanwhile, a *receiver distribution* p_R, is created by convolving the output distribution with the same noise.
4. **KL divergence minimization:** The loss function is the KL divergence from the sender distribution p_S to the receiver distribution p_R. The neural network weights, $\boldsymbol{\omega}$, are updated by stochastic gradient descent to minimize this loss.

This training procedure shows that if no noise is added to p_O, the network will learn to collapse p_O onto $\boldsymbol{x}$. With noise added, however, the network learns to give a probability to all $\boldsymbol{x}'$ relative to how likely they were to produce p_R. The level of noise added is determined by an accuracy schedule, $\{\alpha_0, \ldots, \alpha_{N-1}\}$. The accuracy starts low and increases over time.

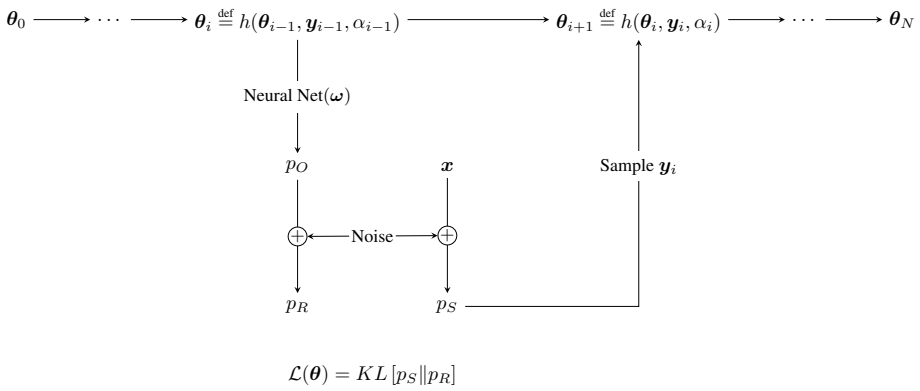

Figure 2: Training process for Bayesian flow networks.

Figure 3 represents the generative process of Bayesian flow networks. The key difference from the training process is that the parameters are updated based on samples from the receiver distribution, rather than the sender distribution. It unfolds as follows:

1. **Initial parameterization:** The process starts with parameters, $\boldsymbol{\theta}_0$, of a prior distribution. For discrete data, these represent uniform probability.
2. **Neural network transformation:** Similar to the training process, the neural network transforms the parameters $\boldsymbol{\theta}_i$ to produce an output distribution p_O.
3. **Noise injection:** Noise is convolved with the output distribution to create the receiver distribution.
4. **Bayesian update:** A sample from the receiver distribution is used to update $\boldsymbol{\theta}_i$ using the Bayesian update function h.
5. **Iterate:** Step 2-4 is repeated N times, after which the parameters are fed into the neural network one last time to produce the final p_O, from which a sample is taken. Note that the noise added to p_O to produce p_R follows the same accuracy schedule as during training.

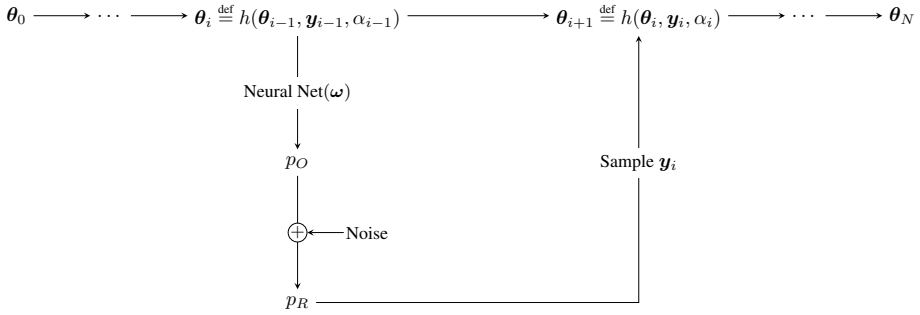

Figure 3: Generative process for Bayesian flow networks.

A comprehensive description of Bayesian Flow Networks is beyond the scope of this paper, but we aim to give the reader a clear understanding of how they differ from diffusion models. We will now look at what $\boldsymbol{\theta}$, h, p_O, p_R, and p_S shown in Figure 2 look like for categorical distributions.

Consider data represented as a D dimensional vector $\mathbf{x} = \left(x^{(1)}, \ldots, x^{(D)}\right) \in \{1, \ldots, A\}^D$, where A is the maximum size of the state space over $\boldsymbol{x}^{(d)}$, and $\{1, \ldots, A\}$ is the set of integers from 1 and A. We will model this as a categorical distribution.

Input distribution The input distribution defines the probability of the data given the parameters fed into the neural network, as shown in Figures 2 and 3. For discrete data, this distribution is modeled as a factorized categorical distribution with parameters $\boldsymbol{\theta} = \left(\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(D)}\right)$, where each $\boldsymbol{\theta}^{(d)}$ includes parameters for a categorical distribution over A categories corresponding to variable d. Specifically, $\theta_a^{(d)}$ represents the probability of category a for the variable d:

$$p_I(\mathbf{x} \mid \boldsymbol{\theta}) = \prod_{d=1}^{D} p_I(\mathbf{x}^{(d)} \mid \boldsymbol{\theta}^{(d)}).$$

Initially, the input distribution is uniform, meaning $\boldsymbol{\theta}_0 = \left[\frac{1}{A}, \ldots, \frac{1}{A}\right]$.

Output distribution $\Psi_{\boldsymbol{\omega}}(\boldsymbol{\theta}_i, i)$ is a neural network model that takes as input a D-dimensional parameter vector $\boldsymbol{\theta}_i$, where each element are parameters of a categorical distribution. The output is of the same type. The output distribution for discrete data is defined based on the data $\mathbf{x}$, model inputs $\boldsymbol{\theta}_i$, step counter i, and resulting model outputs $\Psi_{\boldsymbol{\omega}}\left(\boldsymbol{\theta}_i, i\right) = \left(\Psi_{\boldsymbol{\omega}}^{(1)}\left(\boldsymbol{\theta}_i, i\right), \ldots, \Psi_{\boldsymbol{\omega}}^{(D)}\left(\boldsymbol{\theta}_i, i\right)\right) \in \mathbb{R}^{A \times D}$. The network inputs $\boldsymbol{\theta}_i$ represents the parameters of the factorized categorical distribution $p_I(\mathbf{x} \mid \boldsymbol{\theta}_i)$, while i serves as an additional input that represents the process time. The output distribution is thus defined as

$$p_O(\mathbf{x} \mid \boldsymbol{\theta}_i, i) = \prod_{d=1}^{D} \Psi_{\boldsymbol{\omega}}^{(d)}(\boldsymbol{\theta}_i, i).$$

Here, $\Psi_{\boldsymbol{\omega}}^{(d)}(\boldsymbol{\theta}_i, i)$ denotes A components of the network output corresponding to the parameters $\left(\theta_1^{(d)}, \ldots, \theta_A^{(d)}\right)$ of the categorical distribution for the d-th observation.

Sender distribution A sample from the sender distribution is used to update the parameters $\boldsymbol{\theta}$. For $\mathbf{y} = \left(y^{(1)}, \ldots, y^{(D)}\right) \in \mathcal{Y}^D$, the sender distribution is defined as

$$p_S\left(\mathbf{y} \mid \mathbf{x}; \alpha\right) = \mathcal{N}\left(\mathbf{y} \mid \alpha\left(A\mathbf{e}_{\mathbf{x}} - \mathbf{1}\right), \alpha A \boldsymbol{I}\right),$$

where $\mathbf{e}_{\mathbf{x}}$ is a unit vector of length A and element $\mathbf{x}$ is 1, also known as a one-hot-encoding. The accuracy of these samples is controlled by an accuracy parameter $\alpha \in \mathbb{R}^+$. When α is low, the samples provide limited information about $\mathbf{x}$. As α increases, the samples become increasingly informative about $\mathbf{x}$. Note that the value of $\boldsymbol{y}_i$ determines the amount of information, not the variance. The reason for this is that $\boldsymbol{y}_i$ will play the role of the logits. The value of $\mathbb{E}(\boldsymbol{y}_i)$ increases with α_i, which means that we get a higher focus on the values where $\mathbf{e}_{\mathbf{x}} = 1$.

Receiver distribution The receiver distribution is defined according to the output distribution p_O, and p_S, this takes the form

$$p_R(\mathbf{y} \mid \boldsymbol{\theta}_i; i, \alpha) = \mathbb{E}_{p_O(\mathbf{x}' \mid \boldsymbol{\theta}_i; i)}\left[p_S\left(\mathbf{y} | \mathbf{x}'; \alpha\right)\right],$$

In essence, this integrates over all $\mathbf{x}' \in \{1, \ldots, A\}^D$, considering the contribution of each possible $\mathbf{x}'$ as weighted by its likelihood under the output distribution $p_O(\mathbf{x}|\boldsymbol{\theta}_i, i)$, effectively combines all potential sender distributions into a single receiver distribution.

The Bayesian update function for discrete data introduced in Figure 1 is given by

$$h\left(\boldsymbol{\theta}_i, \mathbf{y}_i\right) = \frac{e^{\mathbf{y}_i} \odot \boldsymbol{\theta}_i}{\sum_{a=1}^{A} e^{\mathbf{y}_{i,a}}\left(\boldsymbol{\theta}_i\right)_a},$$

where $\odot$ refers to the Hadamard product. For a detailed derivation of the update function, refer to the original work by Graves et al. [2023].

At each step, the objective is to minimize the KL divergence from the sender distribution to the receiver distribution over all variables. As the noise in p_S approaches zero, the process is driven to the distribution that maximizes the likelihood of sampling data from the distribution p_O. Graves et al. [2023] show that the loss function for an n-step procedure at step i is:

$$L^N(\boldsymbol{\omega}; \mathbf{x}, \mathbf{y}_i, \boldsymbol{\theta}_i, i) = N\,\mathbb{E}\Bigg[\ln \mathcal{N}\left(\mathbf{y}_i \mid \alpha_i(A\mathbf{e}_{\mathbf{x}} - 1), \alpha_i A I\right) - \ln\left(\sum_{a=1}^{A} p_O\left(a \mid \boldsymbol{\theta}_i, i\right) \mathcal{N}\left(\mathbf{y}_i \mid \alpha_i(A\mathbf{e}_a - 1), \alpha_i A I\right)\right)\Bigg].$$

Furthermore, Graves et al. [2023] shows that if you let N go to infinity you get a continuous-time loss function. In that case, the generative process is not bound by N used during training. We omit the detailed derivation here, but Graves et al. [2023] show that the continuous-time loss works marginally better than the discrete-time loss when N is large. We will use the continuous-time loss in our experiments.

Algorithm 1 Generating Discrete Samples With Bayesian Flow Networks

Require: $\beta(1) \in \mathbb{R}^+$, number of steps $n \in \mathbb{N}$, number of categories $K \in \mathbb{N}$

$\boldsymbol{\theta} \leftarrow \left(\frac{\mathbf{1}}{K}\right)$
for $i = 1$ to n **do**
  $\boldsymbol{k} \sim p_O(\cdot \mid \boldsymbol{\theta}, i)$
  $\alpha \leftarrow \beta(1)(\frac{2i-1}{n^2})$
  $\boldsymbol{y} \sim \mathcal{N}(\alpha(K\boldsymbol{e_k} - 1), \alpha K\mathbf{I})$
  $\boldsymbol{\theta}' \leftarrow e^{\boldsymbol{y}} \odot \boldsymbol{\theta}$
  $\boldsymbol{\theta} \leftarrow \frac{\boldsymbol{\theta}'}{\sum_k \theta'_k}$
end for
$\boldsymbol{k} \sim p_O(\cdot \mid \boldsymbol{\theta}, 1)$

3 Related Work

Several recent works have considered offline RL as a sequence modeling problem. Here we will consider the two most prominent.

3.1 Diffuser

Diffuser [Janner et al., 2022] models state-action trajectories using an unconditional diffusion model. At test time, the current state is imposed through inpainting, while gradients from a differentiable reward model guide sampling toward high-return trajectories. Terminal states can likewise be imposed through inpainting to obtain goal-conditioned plans. The authors further show that the inpainting technique can be used to condition desired end states, effectively allowing the model to solve planning problems it has not been specifically trained for.

3.2 Decision Diffuser

The Decision Diffuser Ajay et al. [2023] differs from Diffuser in two main ways: how it models actions, and how it conditions on rewards. First, Decision Diffuser leverages an inverse dynamics model to capture the relationship between states and actions. This inverse dynamics model estimates actions conditioned on states, effectively predicting the action that brought the environment from one state to the next. This lets the diffusion model focus only on state sequences, rather than on sequences containing both state and action. The authors show empirically that using an inverse dynamics model is advantageous in deterministic environments, but that the performance reduces to the same level as the Diffuser as more stochasticity is introduced into the environment. Second, the Decision Diffuser conditions on return-to-go in a classifier-free manner. This means that the required return is fed into the model during training so that the model learns which sequences to associate with that return. The desired return can again be fed into the model at test time when generating a sequence of future states.

Native categorical diffusion models provide an alternative route to discrete generation [Austin et al., 2021, Lou et al., 2023]. In the discrete experiments below, we therefore compare the categorical BFN with a uniform-categorical diffusion model that predicts the clean trajectory and uses the exact reverse posterior.

4 Method

We propose a sequence-generating approach to reinforcement learning based on Bayesian flow networks capable of planning in both discrete and continuous domains. From now on we will refer to our method as BFN-RL. Like the Decision Diffuser [Ajay et al., 2023], BFN-RL only models sequences of *states*. Return is supplied as a network condition, whereas the current state is fixed by inpainting during sampling. We then utilize a second inverse-dynamics network to model the actions conditioned on the states. In the context of diffusion models, this approach has been shown to provide superior performance compared to modeling state-action pairs from a single model [Janner et al., 2022]. We expect the same benefits when using BFNs as the generative model, but consider further examination of this hypothesis as future work.

In our method, the BFN is trained to generate sequences conditioned on the *return*. The return is the sum of all discounted future rewards and is therefore a measure of the quality of a sequence. The network learns to model the distribution over sequences with both high and low returns. At test time, the desired return biases generation toward trajectories with the corresponding cumulative reward.

4.1 Condition on Return

There are two obvious ways we can condition on return: Either as done by the Diffuser [Janner et al., 2022] or as done by the Decision Diffuser [Ajay et al., 2023]. Considering that the latter option is significantly easier to implement, showed better performance, and does not involve training an extra model, we opted to condition directly on return in a classifier-free manner as was done in Decision Diffuser [Ajay et al., 2023]. At each BFN step, the neural network receives the current parameters of the state distribution, factorized over trajectory time, together with the return and BFN time. The observed current state is not supplied through a separate direct-conditioning pathway.

By conditioning directly on return during the generation process, the model learns to generate trajectories associated with different levels of cumulative reward. At inference time, this allows the generation process to be biased toward higher-return trajectories by conditioning on a desired return value. Furthermore, adopting the Decision Diffuser-style conditioning framework enables straightforward integration with existing sequence-modeling architectures, including the temporal U-net architecture used by Ajay et al. [2023].

4.2 Conditioning on Current State

When Diffuser [Janner et al., 2022] and Decision Diffuser [Ajay et al., 2023] condition on the current state, they both apply an inpainting technique specific to diffusion models, see Eq. (3). During the reverse diffusion process, the part of the sequence that is known (the first state), is replaced by the true value diffused to the appropriate amount for that step. We have adopted a similar technique for BFNs. For discrete data, we set the probabilities of the categorical distribution of the known parts of the object to the appropriate probability for that BFN step. Algorithm 2 shows this method implemented for discrete data. The alterations to BFN sampling (Algorithm 9 in Graves et al. [2023]) are shown in Algorithm 2. For continuous data, the Bayesian update at each step is made similarly, see Algorithm 3 in the Appendix.

The operations involving the mask and conditioning values are those added to the regular BFN algorithm. In Algorithm 2, the mask $\boldsymbol{m}$ selects the observed variables, and $\boldsymbol{c}$ supplies their clean categorical values. All reported experiments use this inpainting mechanism to fix the current state throughout sampling.

For the categorical experiments, we retain the quadratic BFN accuracy schedule $\beta(t) = \beta(1)t^2$, but parameterize its endpoint by $C = K\beta(1)$, where K is the number of categories. This keeps the terminal correct-versus-incorrect message-mean separation from changing merely because a representation uses more categories. We use $C = 36$, 18, and 72 for Empty-Random, DoorKey, and BlockedUnlockPickup, respectively. The continuous-time BFN objective is augmented by clean-data cross entropy with weight 0.5. Reported BFN results use the categorical sampling procedure in Algorithm 2, with 12 updates for Empty-Random and 16 for DoorKey and BlockedUnlockPickup. For categorical tasks, we optimize $\mathcal{L}_{\mathrm{BFN}} + 0.5\mathcal{L}_{\mathrm{CE}}$, where $\mathcal{L}_{\mathrm{CE}}$ is the cross-entropy for predicting the clean category from the BFN input parameters.

Algorithm 2 Inpainting Conditioning for Discrete Random Variables

Require: $\beta(1) \in \mathbb{R}^+$, number of steps $n \in \mathbb{N}$, number of categories K, mask $\boldsymbol{m}$, condition $\boldsymbol{c}$
$\boldsymbol{\theta} \leftarrow \frac{\mathbf{1}}{K}$
for $i = 1$ to n **do**
  $t \leftarrow \frac{i-1}{n}$
  $\mathbf{k} \sim$ DISCRETE_OUTPUT_DISTRIBUTION$(\boldsymbol{\theta}, t)$
  $\alpha \leftarrow \beta(1)\left(\frac{2i-1}{n^2}\right)$
  $\mathbf{y} \sim \mathcal{N}\left(\alpha(K\mathbf{e_k} - \mathbf{1}), \alpha K \boldsymbol{I}\right)$
  $\mathbf{y_c} \leftarrow \alpha(K\mathbf{e_c} - \mathbf{1})$
  $\mathbf{y} \leftarrow \boldsymbol{m} \odot \mathbf{y_c} + (1 - \boldsymbol{m}) \odot \mathbf{y}$
  $\boldsymbol{\theta}' \leftarrow e^{\mathbf{y}} \odot \boldsymbol{\theta}$
  $\boldsymbol{\theta} = \frac{\boldsymbol{\theta}'}{\sum_k \boldsymbol{\theta}'_k}$
end for
$\mathbf{k} \sim$ DISCRETE_OUTPUT_DISTRIBUTION$(\boldsymbol{\theta}, 1)$

5 Experiments

We evaluate our method on two sets of tasks, one with discrete action and state space, and one with continuous action and state space. In the discrete case, we use a grid world environment. For the continuous case, we use the D4RL [Fu et al., 2020] datasets with the Gym-Mujoco suite of environments. We compare our method to the Decision Diffuser [Ajay et al., 2023] and other state of the art offline RL methods. The discrete experiments test whether a categorical BFN can generate executable state plans without hand-coded transition rules. The continuous experiments test whether the same planning formulation remains competitive on standard offline RL benchmarks.

5.1 Discrete Experiments

We evaluate BFN-RL on three MiniGrid tasks, as well as FrozenLake and Sokoban. In each environment, a categorical BFN generates state sequences and a learned inverse-dynamics model maps consecutive states to actions from the environment's native categorical action space. The MiniGrid environments have seven possible actions. We use the same temporal U-Net family across all discrete tasks, with task-specific widths, planning horizons, state representations, and conditioning inputs.

Empty-Random requires navigation to a randomly located goal. DoorKey additionally requires the agent to collect a key and open a door, while BlockedUnlockPickup requires moving an obstructing object before collecting a target box. Their training sets contain 2,000, 5,000, and 3,000 successful trajectories, respectively. The DoorKey and BlockedUnlockPickup datasets include random perturbations followed by expert recovery. We additionally evaluate slippery FrozenLake-8×8 navigation and two-box Sokoban-7×7 using 30,000 and 10,000 successful trajectories, respectively.

At each replanning step, the observed state is fixed through inpainting and one trajectory is sampled. Empty-Random uses a planning horizon of 16 and replans after every action. DoorKey and BlockedUnlockPickup use a horizon of 32 and execute at most 16 actions from each sampled trajectory, replanning earlier if the observed state departs from the generated plan. FrozenLake and Sokoban use horizons of 64 and 32, respectively, and replan after every action. No hand-coded transition projection or analytic inverse dynamics is used.

The MiniGrid experiments condition generation on return, whereas FrozenLake and Sokoban use goal conditioning. To isolate the effect of the generative model, the categorical-diffusion baseline is matched to BFN-RL in training data, temporal U-Net architecture, state representation, conditioning signal, current-state inpainting, inverse-dynamics model, planning horizon, action-execution protocol, and evaluation seeds. The baseline uses uniform categorical corruption with a cosine signal-survival schedule.

The diffusion baseline reaches the goal more quickly in Empty-Random, while BFN-RL has the higher return in DoorKey. On BlockedUnlockPickup, variation between training seeds is larger than the difference in mean success. Overall, BFN-RL is a competitive categorical planner, although it does not uniformly outperform categorical diffusion.

BFN-RL and categorical diffusion perform similarly on FrozenLake. On Sokoban, BFN-RL has higher success for each of the three training seeds and improves the mean success rate by 9.7 percentage points. This identifies a harder discrete task on which the BFN generator provides a consistent advantage under the matched protocol.

Task	Generator	Success (%)	Mean return
Empty-Random-6×6	Categorical diffusion	**100.0 ± 0.0**	**0.941 ± 0.012**
	BFN-RL	99.7 ± 0.6	0.895 ± 0.004
DoorKey-8×8	Categorical diffusion	99.3 ± 0.6	0.916 ± 0.010
	BFN-RL	**100.0 ± 0.0**	**0.934 ± 0.002**
BlockedUnlockPickup	Categorical diffusion	37.0 ± 13.5	0.305 ± 0.113
	BFN-RL	32.7 ± 8.3	0.279 ± 0.072
FrozenLake-8×8	Categorical diffusion	85.5 ± 1.0	
	BFN-RL	**86.7 ± 1.3**	
Sokoban-7×7	Categorical diffusion	59.0 ± 2.6	
	BFN-RL	**68.7 ± 2.5**	

Table 1: Matched discrete-generator comparison. Entries are mean ± sample standard deviation across three training seeds. Evaluation uses 100 episodes per seed except for FrozenLake, which uses 200. Environment and planner seeds are matched between generators.

5.2 Continuous Control

The original Gaussian formulations of Diffuser [Janner et al., 2022] and Decision Diffuser [Ajay et al., 2023] are not natively categorical. We evaluate BFN-RL on D4RL MuJoCo to test whether the same planning framework remains competitive in continuous control.

Table 2 shows the performance of BFN-RL compared to state-of-the-art algorithms. The table shows that BFN-RL is competitive on most datasets.

Dataset	**Environment**	**BC**	**CQL**	**IQL**	**DT**	**TT**	**MOReL**	**Diffuser**	**DD**	**BFN-RL**
Med-Expert	HalfCheetah	55.2	91.6	86.7	86.8	95	53.3	79.8	90.6	97.2 ± 0.1
Med-Expert	Hopper	52.5	105.4	91.5	107.6	110.0	108.7	107.2	111.8	110.3 ± 0.8
Med-Expert	Walker2d	107.5	108.8	109.6	108.1	101.9	95.6	108.4	108.8	106.6 ± 4.9
Medium	HalfCheetah	42.6	44.0	47.4	42.6	46.9	42.1	44.2	49.1	47.8 ± 0.1
Medium	Hopper	52.9	58.5	66.3	67.6	61.1	95.4	58.5	79.3	72.2 ± 7.1
Medium	Walker2d	75.3	72.5	78.3	74.0	79	77.8	79.7	82.5	66.9 ± 5.5
Med-Replay	HalfCheetah	36.6	45.5	44.2	36.6	41.9	40.2	42.2	39.3	42.5 ± 0.5
Med-Replay	Hopper	18.1	95	94.7	82.7	91.5	93.6	96.8	100	91.2 ± 6.3
Med-Replay	Walker2d	26.0	77.2	73.9	66.6	82.6	49.8	61.2	75	58.5 ± 4.8
Average		51.9	77.6	77	74.7	78.9	72.9	75.3	81.8	77.0

Table 2: The table summarizes the test performance of BFN-RL and various other methods for continuous control. The results indicate that BFN-RL can perform at a level comparable to the state of the art. We report mean and standard error over 3 random seeds. All numbers except for BFN-RL are from Ajay et al. [2023].

6 Conclusion

In this work, we introduced a novel approach to reinforcement learning that leverages Bayesian flow networks [Graves et al., 2023] for sequence generation. Our method is capable of planning in both discrete and continuous domains. This shared planning framework also suggests a natural extension to joint continuous-categorical environments, in which each trajectory variable is assigned the corresponding continuous or categorical BFN distribution within one model. The current state is fixed by inpainting throughout sampling, while return conditioning avoids the need for an additional return classifier.

Across the discrete tasks, BFN-RL is competitive with matched categorical diffusion and is consistently stronger on Sokoban. In continuous control, it remains competitive with established offline trajectory models. Together, these results support BFNs as a common planning framework for categorical and continuous domains.

Independent work on Guided-BFNs reports strong results in continuous trajectory-planning environments by supplementing conditional guidance with gradients from a learned reward model during sampling [Li et al., 2024]. That work

considers continuous state–action trajectories, whereas our categorical experiments address planning with discrete states and actions.

Declarations

This research was funded by internal funding from the Norwegian University of Science and Technology.

Appendix A Hyperparameters

Here we present hyperparameters used in the experiments. For the discrete experiments, we used:

- Quadratic accuracy schedule $\beta(t) = \beta(1)t^2$, with $K\beta(1) = 36$, 18, and 72 for Empty-Random, DoorKey, and BlockedUnlockPickup, respectively. The planning horizons are 16, 32, and 32.
- Learning rate 3×10^{-4}, batch size 128, Adam optimizer [Kingma and Ba, 2014], linear warmup for 200 updates on Empty-Random and 300 updates otherwise, followed by cosine decay.
- A generic temporal U-Net with base width 32 for Empty-Random, 48 for DoorKey, and 48 or 96 in the BlockedUnlockPickup sweep.
- A learned inverse-dynamics MLP with two hidden layers of 256 units.
- The continuous-time categorical BFN objective is augmented by a cross-entropy term with weight 0.5.
- Categorical BFN sampling uses 12 updates for Empty-Random and 16 for DoorKey and BlockedUnlockPickup, temperature 0.7 for Empty-Random and 0.9 otherwise, and one generated candidate.
- Empty-Random replans after each action. For DoorKey and BlockedUnlockPickup, at most 16 actions are executed from a plan, with immediate replanning after a state disagreement.
- The matched categorical-diffusion baseline uses cosine uniform corruption, clean-state cross entropy, and 12 or 16 deterministic reverse updates, matching the BFN update count for each task.
- FrozenLake and Sokoban use horizons of 64 and 32, base widths of 64 and 96, batch sizes of 256 and 64, and $K\beta(1) = 72$. Both use cross-entropy weight 0.5, temperature 0.9, 16 sampling updates, and one-action replanning.

Most hyperparameters and model architectures for continuous experiments that are not specific to Bayesian Flow Networks are similar to those used in the official Decision Diffuser implementation.

Hyperparameters used for the continuous experiments:

- The Inverse dynamics model is an MLP with two layers with 256 units and ReLU activations.
- ϵ_θ and f_ϕ are trained for 2×10^6 steps using the Adam optimiser [Kingma and Ba, 2014] with a batch size of 256, a learning rate of 5×10^{-5}.
- We use a planning horizon H of 20.
- For testing we used an exponential moving average of the weights with decay $\alpha = 0.9995$
- First-order BFN solver of Xue et al. [2024] using 10 sampling steps.
- Guidance weight 1.2.
- Temperature 0.5.
- $\sigma_1 = 0.01$.

Appendix B Algorithms

Algorithm 3 gives the continuous-variable counterpart of the inpainting procedure in Algorithm 2.

Algorithm 3 Inpainting Conditioning for Continuous Random Variables

Require: $\sigma_1 \in \mathbb{R}^+$, number of steps $n \in \mathbb{N}$, mask $\boldsymbol{m}$, condition $\boldsymbol{c}$

$\boldsymbol{\mu} \leftarrow \mathbf{0}$
$\rho \leftarrow 1$
for $i = 1$ to n **do**
  $t \leftarrow \frac{i-1}{n}$
  $\hat{\mathbf{x}}(\boldsymbol{\theta}, t) \leftarrow$ CTS_OUTPUT_DISTRIBUTION$(\boldsymbol{\mu}, t, 1 - \sigma_1^2)$
  $\alpha \leftarrow \sigma_1^{-2i/n}\left(1 - \sigma_1^{2/n}\right)$
  $\mathbf{y} \sim \mathcal{N}\left(\hat{\mathbf{x}}(\boldsymbol{\theta}, t), \alpha^{-1}\boldsymbol{I}\right)$
  $\mathbf{y_c} \leftarrow (1 - \sigma_1^{2t})\boldsymbol{c}$
  $\mathbf{y} \leftarrow \boldsymbol{m} \odot \mathbf{y_c} + (\mathbf{1} - \boldsymbol{m}) \odot \mathbf{y}$
  $\boldsymbol{\mu} \leftarrow \frac{\rho\boldsymbol{\mu} + \alpha\mathbf{y}}{\rho + \alpha}$
  $\rho \leftarrow \rho + \alpha$
end for
$\hat{\boldsymbol{x}}(\boldsymbol{\theta}, 1) \leftarrow$ CTS_OUTPUT_DISTRIBUTION$(\boldsymbol{\mu}, 1, 1 - \sigma_1^2)$

Here ρ is the precision of the continuous BFN input distribution; it starts at the unit-precision prior and accumulates the message precisions α.

References

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 119, pages 104–114. PMLR, 2020. URL https://proceedings.mlr.press/v119/agarwal20c.html.

Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B. Tenenbaum, Tommi S. Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision making? In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=sP1fo2K9DFG.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. *arXiv preprint arXiv:2107.03006*, 2021. URL https://arxiv.org/abs/2107.03006.

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 (NeurIPS)*, volume 34, pages 15084–15097, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/7f489f642a0ddb10272b5c31057f0663-Abstract.html.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat GANs on image synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 8780–8794, 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/hash/49ad23d1ec9fa4bd8d77d02681df5cfa-Abstract.html.

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. URL https://arxiv.org/abs/2004.07219.

Alex Graves, Rupesh Kumar Srivastava, Timothy Atkinson, and Faustino Gomez. Bayesian flow networks. *arXiv preprint arXiv:2308.07037*, 2023. URL https://arxiv.org/abs/2308.07037.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS Workshop on Deep Generative Models and Downstream Applications*, 2021. URL https://openreview.net/forum?id=qw8AKxfYbI.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 6840–6851, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. Video diffusion models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 8633–8646, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/39235c56aef13fb05a6adc95eb9d8d66-Abstract-Conference.html.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34,

pages 1273–1286, 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/hash/099fe6b0b444c23836c4a5d07346082b-Abstract.html.

Michael Janner, Yilun Du, Joshua B. Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning (ICML)*, pages 9902–9915, 2022. URL https://proceedings.mlr.press/v162/janner22a.html.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. URL https://arxiv.org/abs/1412.6980.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1179–1191, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/0d2b2061826a5df3221116a5085a6052-Abstract.html.

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. URL https://arxiv.org/abs/2005.01643.

Keyu Li, Zhijie Deng, and Dequan Wang. Guided-BFNs: Towards visualizing and understanding bayesian flow networks in the context of trajectory planning. *Unpublished*, 2024. URL https://openreview.net/forum?id=JRXcvEg3OB.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023. URL https://arxiv.org/abs/2310.16834.

Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. RePaint: Inpainting using denoising diffusion probabilistic models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11461–11471, 2022. URL https://openaccess.thecvf.com/content/CVPR2022/html/Lugmayr_RePaint_Inpainting_Using_Denoising_Diffusion_Probabilistic_Models_CVPR_2022_paper.html.

Shitong Luo and Wei Hu. Diffusion probabilistic models for 3d point cloud generation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2837–2845, 2021. URL https://openaccess.thecvf.com/content/CVPR2021/html/Luo_Diffusion_Probabilistic_Models_for_3D_Point_Cloud_Generation_CVPR_2021_paper.html.

Jan Peters, Katharina Mulling, and Yasemin Altun. Relative entropy policy search. In *AAAI Conference on Artificial Intelligence*, volume 24, pages 1607–1612, 2010. URL https://ojs.aaai.org/index.php/AAAI/article/view/7727.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Kaiwen Xue, Yuhao Zhou, Shen Nie, Xu Min, Xiaolu Zhang, Jun Zhou, and Chongxuan Li. Unifying bayesian flow networks and diffusion models through stochastic differential equations. In *International Conference on Machine Learning (ICML)*, volume 235, pages 55656–55681. PMLR, 2024. URL https://proceedings.mlr.press/v235/xue24d.html.