# Latent Skill Models for Offline Reinforcement Learning

Siddarth Venkatraman

CMU-RI-TR-23-26

May 21, 2023


The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

**Thesis Committee:**
Jeff Schneider, *advisor*
David Held
Lili Chen

*Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.*

*To those who supported me, especially my parents and brother.*

# Abstract

Offline reinforcement learning (RL) holds promise as a means to learn high-value policies from a static dataset, without the need for further environment interactions. However, a key challenge in offline RL lies in effectively stitching portions of suboptimal trajectories from the static dataset while avoiding extrapolation errors arising due to a lack of support in the dataset. Existing approaches use conservative objectives that favor pessimistic value functions or rely on generative modelling with noisy Monte Carlo return-to-go samples for reward conditioning. The key challenge in offline RL is identifying the behavioral primitives (a.k.a skills) that exist in the offline dataset, and chain these behaviors together to produce high-value policies.

In this thesis, we investigate latent variable models having different levels of expressiveness to model skills as compressed latent vectors, and then compose these skills to solve a specific task. We first describe a Variational Autoencoder (VAE) based method which learns a temporally abstract world model that predicts the state outcome of executing a skill, and then uses this model to do Online Planning with Offline Skill Models (**OPOSM**). We then extend this method to instead work with a Vector-Quantized VAE to learn a bank of discrete latents skills (**VQSkills**). Finally we investigate using latent diffusion models to learn a multimodal skill prior, and then use this prior to perform batch constrained Q-learning. We call this algorithm Latent Diffusion Constrained Q-Learning (**LDCQ**). We empirically demonstrate the effectiveness of these algorithms to learn high-value policies in the D4RL benchmark suite.

# Acknowledgments

I would like to begin by thanking express my advisor and committee chair, Professor Jeff Schneider, for his constant support and great research feedback throughout my Masters study at CMU. I would like to thank everyone who helps keep the Auton Lab running, especially Dr. Predrag Punosevac for his vigilant maintenance and support of the lab compute infrastructure.

I am grateful to have had the pleasure to collaborate with several brilliant researchers these last two years. I would like to thank Benjamin Freed whom I collaborated with for OPOSM described in this thesis. He led the research project, and was wonderful to work with. I am also grateful to Shivesh Khaitan and Ravi Tej Akella with whom I worked on Latent Diffusion Reinforcement Learning, also described in this thesis. I am thankful to other PhD students under Prof. Schneider with whom I have had many fun and creatively stimulating conversations. I would give special mention to those who part of the self-drive discussions, namely Adam Villaflor, Brian Yang and Yeeho Song. The insightful technical and non-technical conversations helped me constantly patch any blind spots, and come up with many interesting ideas.

Looking further back, I would never have gotten this far without my wonderful colleagues at the robotics team Project MANAS in my undergraduate. It was through this project that I discovered my passion for the fields of machine learning and robotics. I dedicate my achievements to my parents and brother, whose unwavering love and support has been invaluable to my mental health and happiness.

# Funding

We thank CMU Argo AI Center and Professor Jeff Schneider for supporting the work presented in this thesis.

# Contents

*When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.*

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Reinforcement Learning (RL) techniques have proven to be very effective at solving complex decision making problems given enough environment interactions during training. RL agents have achieved superhuman performance in complex games such as Go (Silver et al. [58], Silver et al. [59]), Chess (Schrittwieser et al. [54]), Atari (Mnih et al. [42]), Dota (Berner et al. [4]) and Starcraft (Vinyals et al. [71]). Reinforcement Learning has also demonstrated great success in producing policies suitable for complex optimization tasks such as prediction of protein folding (Jumper et al. [30]), control of tokamaks for nuclear fusion (Degrave et al. [7]) and finding efficient tensor factorization for matrix multiplication (Fawzi et al. [10]). Despite these successes, RL has remained difficult to apply in real world robotics control settings. A major reason for this is the sample complexity of popular RL algorithms, and the difficulty with obtaining the required amount of data in any real world control setting. In domains like self driving for example, it is prohibitively expensive, unsafe, and slow to deploy an agent on real roads to collect the necessary data to learn a high value driving policy.

Offline Reinforcement Learning (Levine et al. [37]) offers a promising approach to learning policies from static datasets. These datasets are often comprised of undirected demonstrations and suboptimal sequences collected using different *behavior policies*. Unlike simple behavioral cloning, offline RL seeks to *improve* upon the behavior policy instead of simply mimicking it. In many tasks such as self driving, while deploying an RL agent to learn from environment interactions is not feasible, there is

a large amount of driving data we can collect from human drivers that we could use to train offline RL policies. These policies can also then be finetuned with online RL training with significantly fewer real world interactions. However, offline RL presents a whole new host of challenges to deal with, as it is the most extreme version of off-policy learning where the agent must learn from demonstrations not under its own policy. Because the agent cannot interact with the environment, policies that stray outside the support of the behavior datasets can have unpredictable behavior. The key challenge in offline RL is to learn policies that improve upon the behavior policy while remaining within states in the dataset support.

At the core of many offline RL algorithms is an attempt to mitigate the *extrapolation error* which arises while querying the learned Q-function on out-of-support samples for policy improvement. In order to extract the best policy from the data, Q-learning uses an *argmax* over actions to obtain the temporal-difference target. However, querying the Q-function on out-of-support state-actions can lead to errors via exploiting an imperfect Q-function (Fujimoto et al. [13]). This leads to compounding bootstrapping error that results in learning a suboptimal policy.

Behavior policies in offline RL data can be highly multimodal due to either being collected from undirected demonstrations, or bevause the demonstrator can solve the task in multiple ways. Human data especially can be very multimodal, and often suboptimal. The offline RL objective in many cases can be reduced to stitching together behavioral primitives from the offline dataset to solve the specified task. We will refer to these behavioral primitives extracted from undirected demonstrations as **skills**. If we can ensure our policy always picks skills that are in the dataset support, we can prevent extrapolation error. To extract these skills from the dataset, recent methods (Pertsch et al. [45], Ajay et al. [2]) have proposed to learn the behavior policy using latent variable models, and to express these skills as compressed latent vectors.

In chapter 3, we first describe a novel algorithm which learns temporally abstract world models to predict the state outcome of executing these latent skills, and then chains together skills to maximize reward in an environment using MPC. In many environments the world model can be generalized easier than value function outside the dataset support, thus mitigating extrapolation error. The use of a temporally abstract world model also simplifies long horizon skill stitching as we will show.

We use a Variational Autoencoder (Kingma and Welling [32]) to learn a latent skill representation. The decoder of the VAE consists of both the low level control policy and the temporally abstract world model. We show that naive optimization of the VAE evidence lower bound objective (ELBO) does not result in learning an approximate posterior that respects the true causal structure of the Markov Decision Process (MDP). We instead propose an Expectation-Maximization (EM) training objective that allows us to learn causally correct skill models with amortized variational inference. We call this algorithm "Online Planning with Offline Skill Models", or **OPOSM**. We then extends OPOSM to learn discrete latent skills by replacing the VAE with a Vector-Quantized Variational Autoencoder (VQ-VAE) (van den Oord et al. [68]) as the skill model. We show that the discrete latent variables can better capture the multimodality present in the offline datasets than a conditional Gaussian distribution used in a standard VAE. We also show that using discrete latent variables can reduce exploitation of the world model during planning. We call this algorithm **VQSkills**.

Next, we investigate further the relationship between the temporal abstraction of the skills and the multimodality in the latent space. We show that more expressive generative models are necessary to leverage more high information latents. Recently, diffusion generative models (Sohl-Dickstein et al. [60], Song and Ermon [62]) have demonstrated remarkable success in complex tasks such as image generation (Ramesh et al. [47]) and neural rendering (Jun and Nichol [31]). Diffusion has been used for offline RL by modelling trajectories in the offline dataset, and planning using classifier guidance (Janner et al. [29]), or return conditioning (Ajay et al. [3]). In chapter 4 we propose a novel algorithm using latent diffusion models to learn for expressive multimodal skill priors, and use this model to learn abstract Q-functions of the form $Q(s, z)$ through batch-constraining. The Batch-constrained Q-Learning (BCQ) (Fujimoto et al. [13]) framework allows learning offline RL policies without introducing pessimism in the value function or difficult to optimize objectives. This allows us to learn Q functions only querying skills within the support of the behavior dataset, preventing extrapolation error. We call this algorithm Latent Diffusion-Constrained Q-Learning, or **LDCQ**.

We evaluate all our algorithms on the D4RL benchmark suite (Fu et al. [11]), which contains several offline RL datasets for tasks in various environments. Our

results are competitive with state-of-the-art (SOTA) offline RL algorithms. Our latent skill algorithms particularly excel at sparse reward tasks which require long horizon reasoning. We find that the performance gains come primarily due to temporal abstraction. We also show empirically in these tasks that improving the generative model from a VAE to an expressive latent diffusion model reduces extrapolation error and thus results in better task performance.

To sum up, in this thesis we propose latent variable skill models of different levels of expressiveness to model behaviors in offline datasets, and use these skill models to create high value policies. We propose learning temporally abstract world models to do MPC with these latent skills to do efficient long horizon planning. We also propose a model-free skill learning algorithm using latent diffusion which achieves SOTA in several challenging offline RL tasks in the D4RL benchmark suite. The proposed algorithm is simple to tune, and scales well with more data since this is directly tied to the expressiveness of the generative model. Diffusion models have been shown to scale extremely well in other generative modelling domains, and by framing offline RL as a generative modelling problem we can leverage these models to learn complex and high-value control policies in domains where suboptimal demonstrator data is cheap and simple to obtain but environment interactions are costly.

# Chapter 2

# Background

## 2.1 Reinforcement Learning

### 2.1.1 Preliminaries

Reinforcement Learning (RL) is a sub-field of machine learning inspired by dopamine-maximization mechanisms studied in behavioral psychology. In the RL framework, the *agent* exists in an *environment*, and takes *actions* in it to obtain *rewards*. This reward is considered to be generated from the environment in response to actions executed by the agent from a certain state. The goal in RL is for the agent to learn through environment interactions to produce actions that maximize the total reward.

We now lay out the concrete mathematical framework for RL which previously described at a high level. In the RL framework, every task has an associated Markov Decision Process (MDP). This MDP is a tuple $\langle \rho_0, \mathcal{S}, \mathcal{A}, r, P, \gamma \rangle$, where $\rho_0$ is the initial state distribution, $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function that defines the probability of moving from one state to another after taking an action, and $\gamma \in [0, 1)$ is the discount factor that determines the importance of future rewards. The goal in RL is to learn a policy distribution $\pi(\mathbf{a}|\mathbf{s}); \mathbf{a} \in \mathcal{A}, \mathbf{s} \in \mathcal{S}$, i.e., a mapping from states to actions, that maximizes the expected cumulative discounted reward $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$.

In deep RL, we parameterize the policy with parameters $\theta$ and represent this as $\pi_\theta(\mathbf{a}|\mathbf{s})$. We represent a state-action trajectory tuple compactly with $\boldsymbol{\tau} =$

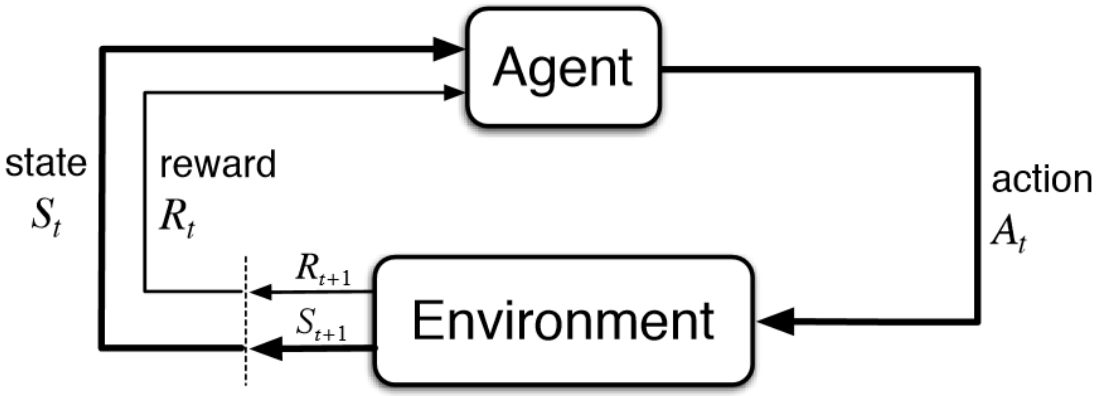



Figure 2.1: **The Reinforcement Learning framework:** The agent uses its policy to generate an action conditioned on its current state. The environment then changes its state based on its transition dynamics, and the agent receives the new state and reward generated by the environment. Figure from Sutton and Barto [66].

$(\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, ...)$. Then we can represent the distribution of trajectories obtained under the policy $\pi_\theta$ as $p_\theta(\boldsymbol{\tau})$. The objective in RL is now to find the optimal policy parameters $\theta^*$ such that:

$$\theta^* = \underset{\theta}{\text{argmax}} \mathbb{E}_{\tau \sim p_\theta(\tau)}[\sum_{t=0}^{\infty} \gamma^t r_t] \tag{2.1}$$

### 2.1.2 Value functions

Value functions are useful tools to learn policies in RL. For a policy $\pi$ there are two types of value functions, the state value function $V^\pi(\mathbf{s})$ and the action value function $Q^\pi(\mathbf{s}, \mathbf{a})$. The V-function is the expected discounted return starting at the state $\mathbf{s}$ under the policy $\pi$, while the Q-function is the expected discounted return starting from state $\mathbf{s}$ and taking action $\mathbf{a}$ and then following the policy $\pi$. There is a simple equation relating these two functions:

$$V^\pi(\mathbf{s}) = \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})}[Q^\pi(\mathbf{s}, \mathbf{a})] \tag{2.2}$$

The optimal value functions denoted as $V^*$ and $Q^*$ are the value functions under the optimal policy $\pi_{\theta^*}$. Given the optimal Q-function, we can get a deterministic optimal policy function $\pi^*(\mathbf{s})$ for any state $\mathbf{s}$ by greedy maximization:

$$\pi^*(\mathbf{s}) = \underset{\mathbf{a}}{\text{argmax}} Q^*(\mathbf{s}, \mathbf{a}) \tag{2.3}$$

Thus, all we need to obtain the optimal policy is the optimal Q-function.

### 2.1.3 Q-learning with Deep Q-Networks

The basis for many RL algorithms is Q-learning. In deep RL where the Q-function is parameterized by a neural network, DQN (Mnih et al. [43]) is the base algorithm for learning the optimal Q-function. In DQN, we collect rollouts under the current policy $\pi(\mathbf{s}) = \underset{\mathbf{a}}{\text{argmax}} Q_\theta(\mathbf{s}, \mathbf{a})$ and then minimize the error between the Q-function prediction $Q_\theta(\mathbf{s}_t, \mathbf{a}_t)$ and its corresponding temporal difference (TD) target $r(\mathbf{s}_t, \mathbf{a}_t) + \gamma Q_\theta(\mathbf{s}_{t+1}, \underset{\mathbf{a}}{\text{argmax}} Q_\theta(\mathbf{s}_{t+1}, \mathbf{a}))$ (Sutton and Barto [65]). To make learning more stable, we maintain a separate target Q-network that lags behind the current Q-network. The base DQN algorithm is detailed in Algorithm 1:

### 2.1.4 Online RL

This is the most straightforward RL setting, where the agent gathers experiences by constantly exploring and exploiting the environment, getting live feedback to update its policy. This can be further divided into on-policy and off-policy algorithms.

In the on-policy setting, all data used for policy updates are obtained under the current policy. This form of RL can be very effective at learning high-value policies but is quite data inefficient, since the agent cannot reuse data collected earlier in training under a previous policy for making new policy updates. Some popular on-policy RL algorithms are REINFORCE (Sutton et al. [67]) and Proximal Policy Optimization (PPO) (Schulman et al. [55]).

Off-policy RL is a paradigm in which an agent can learn from data generated by a different policy than the one being updated. For online off-policy algorithms, the agent can interact with environment to collect new experiences as well, but stores previously collected data into a replay buffer for future policy updates. Algorithms

**Algorithm 1** Deep Q-Network (DQN)

**Input:** Initialize replay buffer $\mathcal{D}$
Initialize action-value function $Q_\theta$ with random weights
Initialize target action-value function $\hat{Q}$ with weights of $Q_\theta$
Set exploration rate $\epsilon$, minibatch size $B$, discount factor $\gamma$, maximum number of episodes $N$, maximum number of steps per episode $T$

**for** episode = 1 to $N$ **do**
  Initialize state $s_1$
  **for** $t = 1$ to $T$ **do**
    With probability $\epsilon$ select a random action $a_t$
    Otherwise, select action $a_t = \arg\max_a Q(s_t, a)$
    Execute action $a_t$, observe reward $r_t$ and new state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $\mathcal{D}$
    Sample random minibatch of transitions $(s_j, a_j, r_j, s_{j+1})$ from $\mathcal{D}$
    Set target for iteration $i$ as

$$y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, \underset{a}{\operatorname{argmax}} \hat{Q}(s_{j+1}, a))) & \text{otherwise} \end{cases}$$

    Perform a gradient descent step on the loss

$$L(\theta) = \left(y_j - Q_\theta(s_j, a_j)\right)^2$$

    Every $C$ steps, reset $\hat{Q} = Q_\theta$
  **end for**
**end for**

that can work with off-policy data are generally significantly more data efficient. Some popular off-policy RL algorithms are DQN (Mnih et al. [43]), Soft Actor-Critic (SAC) (Haarnoja et al. [16]) and Twin Delayed DDPG (TD3) (Fujimoto et al. [12]).

### 2.1.5 Offline RL

Offline RL (Levine et al. [37]) focuses on learning from a fixed dataset of pre-collected experiences from a behavior policy without further interaction with the environment. In offline RL, the data is typically obtained from an external source, such as human demonstrations or historical data. The agent learns from this fixed dataset to improve its policy without any additional interaction with the environment. Offline RL is

particularly useful when online data collection is costly, dangerous, or slow. In this thesis, we focus on the offline RL setting.

In offline RL, the agent has access to a static dataset $\mathcal{D} = \{\mathbf{s}_t^i, \mathbf{a}_t^i, \mathbf{s}_{t+1}^i, r_t^i\}$ of transitions generated by a unknown behavior policy $\pi_\beta(\mathbf{a} \mid \mathbf{s})$ and the goal is to learn a new policy using only this dataset without interacting with the environment. Unlike simple behavioral cloning, offline RL methods seek to improve upon the behavior policy used to collect the offline dataset. The distribution mismatch between the behavior policy and the training policy can result in problems such as querying the target Q-function with actions not supported in the offline dataset leading to an optimism bias. This bias results in compounding error during Q-learning, and the lack of new environmental interactions means these errors cannot be corrected with feedback. This is called **extrapolation error**. At the goal of many offline RL algorithms is mitigating the extrapolation error during Q-learning. We now discuss some important offline RL algorithm classes.

### Conservative methods

These methods alter the learning objective to try to keep the target policy close the behavior policy, increasing conservatism. A popular offline RL algorithm of this type is Conservative Q-Learning (CQL) (Kumar et al. [36]), which augments the Q-Learning objective to encourage a pessimistic bias to the Q-function outside the behavior support:

$$L(\theta) = \alpha(\mathbb{E}_{s\sim\mathcal{D},a\sim\mu(a|s)}[Q_\theta(s,a)] - \mathbb{E}_{s\sim\mathcal{D},a\sim\hat{\pi}_\beta(a|s)}[Q_\theta(s,a)]) + ||TDError||_2^2 \quad (2.4)$$

The objective reduces Q-values for state-actions sampled by the current policy $\mu$ and raises Q-values for state-actions drawn from the behavior support. The $\alpha$ multiplier controls the trade-off between conservatism and policy improvement.

### Batch-constrained methods

In batch-constrained Q-learning (BCQ), the target Q-function is constrained to only be maximized using actions that were taken by the demonstrator from the given state

(Fujimoto et al. [13]).

$$\pi(\mathbf{s}) = \underset{\substack{\mathbf{a} \\ s.t.(\mathbf{s},\mathbf{a})\in\mathcal{D}}}{\operatorname{argmax}} Q(\mathbf{s}, \mathbf{a}) \tag{2.5}$$

In a deterministic MDP setting, BCQ is theoretically guaranteed to converge to the optimal batch-constrained policy. In any non-trivial setting, constraining the policy to actions having support from a given state in the dataset is not feasible, especially if the states are continuous. Instead, a function of the form $\pi_\psi(\mathbf{a} \mid \mathbf{s})$ must be learned on the demonstrator data and samples from this model are used as candidates for the argmax:

$$\pi(\mathbf{s}) = \underset{\mathbf{a}_i\sim\pi_\psi(\mathbf{a}|\mathbf{s})}{\operatorname{argmax}} Q(\mathbf{s}, \mathbf{a}_i) \tag{2.6}$$

The BCQ framework is attractive because the quality of the learnt policy is largely dependent on the expressiveness of the generative model, and its fit to the behavior policy. In our LDCQ method described in chapter 4, we leverage latent diffusion models to do constrained Q-learning.

Some other algorithms such as Implicit Q-Learning (IQL) (Kostrikov et al. [34]) also try to learn a batch-constrained policy, but do so without the need for a generative model to sample actions. In IQL's case, it does so by using an expectile regression loss to do a tradeoff between DQN and SARSA. This is dependent on the expectile parameter $\tau$, and as $\tau \to 1$ the policy converges to the optimal batch constrained policy. However, raising $\tau$ closer to 1 makes the optimization problem more challenging.

### 2.1.6 Model-Free RL

Model-Free RL methods directly learn a policy or value function from interactions with the environment without explicitly building a model of the environment. These methods make decisions based on trial and error and learn from the observed rewards and states. Some popular Model-Free RL algorithms are DQN, PPO and SAC.

### 2.1.7 Model-Based RL

Model-based RL (MBRL) methods involve building an explicit model of the environment. These models capture the transition dynamics of the environment, allowing the agent to plan ahead and simulate possible trajectories before taking actions. The

model can be a learned model or a known model of the environment. By leveraging the learned model, the agent can optimize its policy more efficiently and reduce the number of interactions with the real environment. Model-based RL methods can achieve faster learning and sample efficiency compared to model-free methods. However, building an accurate model can be challenging, and errors in the model can lead to suboptimal policies. In some algorithms, the model is used for planning at test time as well. Some popular model based RL algorithms are MuZero (Schrittwieser et al. [54]), Dreamer (Hafner et al. [17]), MBPO (Janner et al. [27]) and PlaNet (Hafner et al. [18]).

## 2.2 Latent Variable Models

Latent variable models are a class of statistical models that aim to capture hidden or unobserved factors, known as latent variables, that influence the observed data. These models are used to represent complex relationships and dependencies among variables in a more compact and interpretable manner. In latent variable models, the observed variables are assumed to be generated from a set of underlying latent variables. The latent variables are not directly observed but are inferred based on the observed data. The idea is that the latent variables capture the underlying structure or sources of variation in the data, which cannot be explained by the observed variables alone.

Many deep generative models are different forms of latent variable models. The structure, representation, number of latent variables, and their causal relationship give us different objectives to optimize, which result in generative models with varying levels of expressiveness. We will now discuss latent variable model classes relevant to this thesis.

### 2.2.1 Variational Autoencoder

Variational Autoencoders (VAE) (Kingma and Welling [32]) are deep generative models which consist of an encoder and decoder network. VAEs try to model the complex multimodal data distribution $p(x)$ by introducing a latent variable $z$ such that $p(x|z)$ is a simple distribution such as a Gaussian. Concretely, we have:

$$p(x) = \int p(x|z)p(z)dz \tag{2.7}$$

Where $p(z)$ is a simple to sample from distribution such as a standard Normal. However, it is difficult to maximize this objective since for any data point $x$, most $z$ samples from $p(z)$ will result in low probability $p(x|z)$. Hence, we instead do importance sampling from the posterior $p(z|x)$:

$$p(x) = \int p(x|z)p(z)\frac{p(z|x)}{p(z|x)} = \mathbb{E}_{z\sim p(z|x)}[\frac{p(x|z)p(z)}{p(z|x)}] \approx \mathbb{E}_{z\sim q_\phi(z|x)}[\frac{p(x|z)p(z)}{q_\phi(z|x)}] \tag{2.8}$$

Where in the final step we replaced the true posterior $p(z|x)$ with an approximate posterior $q_\phi(z|x)$ which is our encoder network, since the true posterior is intractable. Taking log on both sides, applying Jensen's inequality and shuffling around some terms, we end up with a lower bound on $p(x)$ which is tight when $D_{KL}(q_\phi(z|x)||p(z|x)) = 0$. This bound is called the ELBO (Evidence Lower BOund), and is the objective we try to maximize. We also parameterize the decoder $p_\theta(x|z)$ and jointly optimize $\theta$ and $\phi$ to maximize the lower bound:

$$\log p(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p(z)) \tag{2.9}$$

The first term is the reconstruction objective and the second term is latent regularization. The reconstruction term when optimized tries to create a latent space which has rich information for the decoder to reconstruct $x$, while the regularization term keeps the latent distribution structured. This regularization is necessary to later generate new samples with the model by sampling $z$ from the standard normal prior. We can control the information contained in the latent variable $z$ by multiplying the regularization term with a hyper-parameter $\beta$:

$$J(\theta, \phi) = \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - \beta D_{KL}(q_\phi(z|x)||p(z)) \tag{2.10}$$

This formulaation is called a $\beta$-Variational Autoencoder (Higgins et al. [20]). Higher values of $\beta$ correspond to a better regularized, disentangled latent space at the cost of reconstruction loss, whereas low $\beta$ values result in high information latents which are not structured to be sampled from the prior distribution.

### 2.2.2 Vector-Quantized Variational Autoencoder

Discrete latent variables are attractive since they can express latent multi-modality quite well, at the cost of increased difficulty in modelling the distribution they are drawn from. VQ-VAEs ((van den Oord et al. [68])) offer a way of learning discrete latent variables using continuous optimization.

In a VQ-VAE, a codebook $Q_{z_{dim}\times k}$ is a matrix of $k$ embedding vectors the size of the latent vectors $z$ which are output by the encoder, each of length $z_{dim}$. During training, the latents(There can be multiple) $z_e$ output by the encoder are compared

against all the vectors of the codebook to find their nearest neighbours, which are then assembled together to pass to the decoder. We call this nearest codebook latent $z_q$. A uniform prior eliminates the KL term from the standard VAE ELBO. The training loss for a VQ-VAE is:

$$L(\theta, \phi, z_q) = \mathbb{E}_{x \sim \mathcal{D}}[||x - p_\theta(z_q)||^2 + ||sg(z_\phi) - z_q||^2 + \beta||z_\phi - sg(z_q)||^2] \tag{2.11}$$

Where $z_\phi$ refers to $z$ output by the encoder $q_\phi$, $z_q$ is the nearest neigbour of $z_\phi$ in the codebook, $p_\theta$ is the decoder, $sg$ is the stop gradient operator. Since the vector quantization (nearest neighbor) step is not differentiable, we can propagate approximate gradients into the encoder by using the straight through gradient trick where instead of directly passing $z_q$ into the decoder, we can pass a differentiable version using $z_q := z_\phi + sg(z_q - z_\phi)$. After jointly training the encoder, codebook and decoder, a discrete prior $p_\omega(z)$ is trained, and we sample from this during data generation at test time.

### 2.2.3 Diffusion Probabilistic Models

Diffusion models (Sohl-Dickstein et al. [60], Song and Ermon [62]) are a class of latent variable generative model which learn to generate samples from a probability distribution $p(\mathbf{x})$ by mapping Gaussian noise to the target distribution through an iterative process. They are of the form $p_\psi(\mathbf{x}_0) := \int p_\psi(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T}$ where $\mathbf{x}_0, \dots \mathbf{x}_T$ are latent variables and the model defines the approximate posterior $q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)$ through a fixed Markov chain which adds Gaussian noise to the data according to a variance schedule $\beta_1, \dots, \beta_T$. This process is called the *forward diffusion process*:

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1}), \qquad q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}) \tag{2.12}$$

The forward distribution can be computed for an arbitrary timestep $t$ in closed form. Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$. Then $q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I})$.

Diffusion models learn to sample from the target distribution $p(\mathbf{x})$ by starting from Gaussian noise $p(\mathbf{x}_T) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and iteratively *denoising* the noise to generate

in-distribution samples. This is defined as the *reverse diffusion process* $p_\psi(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$:

$$p_\psi(\mathbf{x}_{0:T}) := p(\mathbf{x}_T) \prod_{t=1}^{T} p_\psi(\mathbf{x}_{t-1} \mid \mathbf{x}_t), \qquad p_\psi(\mathbf{x}_{t-1} \mid \mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \mu_\psi(\mathbf{x}_t, t), \mathbf{\Sigma}_\psi(\mathbf{x}_t, t)) \tag{2.13}$$

The reverse process is trained by minimizing a surrogate loss-function (Ho et al. [22]):

$$\mathcal{L}(\psi) = \mathbb{E}_{t\sim[1,T],\mathbf{x}_0\sim q(\mathbf{x}_0),\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})} \mid\mid \epsilon - \epsilon_\psi(\mathbf{x}_t, t) \mid\mid^2 \tag{2.14}$$

Diffusion can be performed in a compressed latent space $\mathbf{z}$ (Rombach et al. [49]) instead of the final high-dimensional output space of $\mathbf{x}$. This separates the reverse diffusion model $p_\psi(\mathbf{z}_{t-1} \mid \mathbf{z}_t)$ from the decoder $p_\theta(\mathbf{x} \mid \mathbf{z})$. The training is done in two stages, where the decoder is jointly trained with an encoder, similar to a $\beta$-Variational Autoencoder with a low $\beta$. The diffusion prior is then trained to fit the optimized latents of this model.

# Chapter 3

# Planning with Latent Skills for Offline RL

## 3.1 Introduction

In recent years, there has been great interest in trying to learn control policies that can act at different levels of hierarchy. For many tasks, low level control is easier when they are used to follow some higher level instruction. Humans routinely form hierarchical plans to break down complex tasks into simpler blocks. In driving for example, we have higher level behaviors like yielding to incoming traffic, changing lanes, overtaking etc. Skill Learning has emerged as a popular area in Reinforcement Learning, which tries to jointly tackle the problem of automatically inferring useful skills from data and then learning to predict the appropriate skill using a high level policy.

Learning temporally abstract skills using variational inference has become a popular method for learning these hierarchical policies. The primary challenge that our approach overcomes is in learning a *causal* skill-conditioned world model, when online experimentation is not possible. We propose learning a Temporally Abstract World Model (TAWM) which predicts the state outcome of executing a skill for a fixed horizon, and then use this model for Model Predictive Control (MPC). In the offline setting, deducing the true causal effect of skills on the state of the environment

is challenging because one cannot repeatedly deploy skills in the environment to observe their effect (Pearl [44]). In fact, we show that a naive VI-based approach similar to that taken by previous methods like OPAL (Ajay et al. [2]), PLAS (Zhou et al. [73]), and SPiRL (Pertsch et al. [45]) are not guaranteed to correctly model these causal relationships, instead leading the agent to overestimate its influence on the environmental state. We propose a principled method to fix this issue with a modified VI approach, in which the approximate posterior is constrained to match the true causal structure of the environment. We call our algorithm Online Planning with Offline Skill Models (**OPOSM**).

We next show that optimization of these continuous skill latent variables using algorithms like CEM during MPC can exploit the neural network dynamics model and find faulty optima that degrade task performance. We show that this problem can be mitigated by expressing the skills using a small number of discrete latents. We propose using vector quantization to learn these discrete skills, inspired by the success of VQ-VAEs for image generative modelling. We demonstrate our offline MBRL approach called **VQSkills** for long horizon planning in the antmaze D4RL environments (Fu et al. [11]), which require long horizon planning with complex dynamics, achieving state of the art results. We also match state of the art performance in the Carla NoCrash autonomous driving benchmark (Codevilla et al. [6]), which we use to illustrate the advantage of vector quantization over continuous skills.

## 3.2 Related Work

There have been a number of algorithms proposed which try to use variational inference in order to build a skill model (Ajay et al. [2], Pertsch et al. [45], Achiam et al. [1], Pertsch et al. [46], Eysenbach et al. [9]). PLAS ([73]) is also similar to these skill learning methods, despite not referring to the latent variables as skills. Most of these skill learning methods are model-free, but a few model based skill approaches have also been proposed (Sharma et al. [56], Sharma et al. [57]). Unlike these, we learn a temporally abstract world model that predict the state $T$ time steps away from the start of skill execution. This allows us to plan long horizon trajectories with fewer optimization variables.

SPLT (Villaflor et al. [70]) learns discrete latents that characterize different behavioral modalities. Since they only have a small number of discrete latents, they can optimize using exhaustive search. If the latents need to be more expressive this would mean increasing the number greatly which is not computationally feasible to optimize. The vector quantized skills proposed in this paper can be optimized with a modified CEM planner which is significantly more efficient. VQ-VAE and its successors (Razavi et al. [48]) are state of the art latent variable models for image generation. Other methods of using discrete latent variables have been proposed, like Gumbel-Softmax[26]. However, these methods are not stable for deep network optimization, and produce worse log-likelihoods.

## 3.3 OPOSM

In this section, we describe our method Online Planning with Offline Skill Models (OPOSM). We begin by explaining and how to train a Temporally Abstract World Model (TAWM) while respecting the causal structure of the MDP. We then describe how to use the TAWM for MPC.

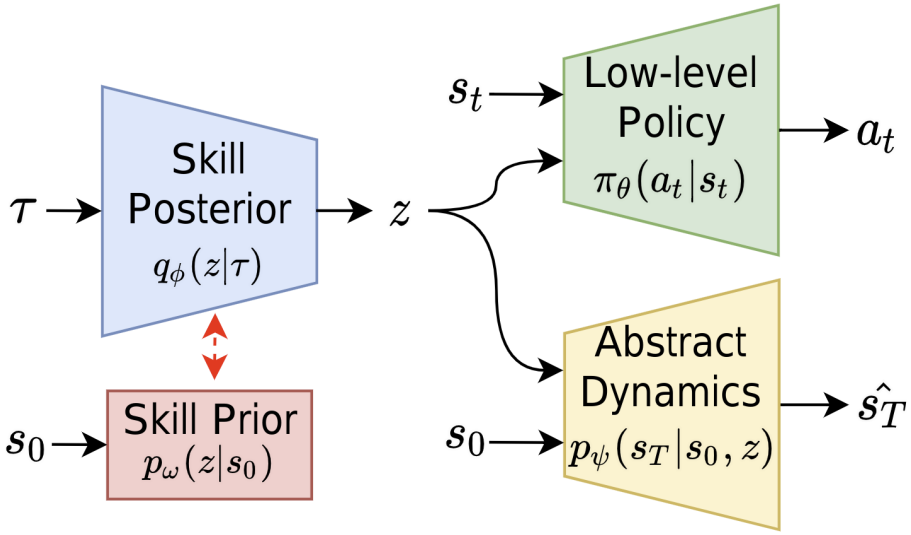



Figure 3.1: **Skill Model and TAWM Overview** The skill posterior/VAE encoder takes as input a sub-trajectory $\tau$ and infers the skill $z$. The skill prior tries to also predict this $z$ but only conditioned on the first state $s_0$. The policy and TAWM are conditioned on the skill, and try to predict the action sequence and future state respectively. $z$ is detached from the computational graph before passing to the TAWM, so that it doesn't propagate gradients to the encoder.

### 3.3.1 Training Skill Model and TAWM

The primary challenge of learning a TAWM from offline data is ensuring that the TAWM accurately captures the true causal influence of skills on long-term state transitions. A naive approach to learning the skill model and the TAWM $p_\psi$ would be to treat skills as latent variables, and optimize the following evidence lower bound

(ELBO):

$$\mathcal{L}(\theta, \psi, \phi, \omega) = \mathbb{E}_{\tau \sim \mathcal{D}}\Big[\mathbb{E}_{q_\phi(z|\tau)}\left[\log \pi_\theta(\vec{a}|\vec{s}, z) + \log p_\psi(s_T|s_0, z)\right] - D_{KL}(q_\phi(z|\tau)||p_\omega(z|s_0))\Big] \tag{3.1}$$

Where $\tau$ represents a tuple of states and actions from a sub-trajectory, $q_\phi$ is the encoder/approximate posterior of the VAE, $\pi_\theta$ is the decoder of the VAE and the low level control policy conditioned on the current state and skill latent $z$, $p_\psi$ is the TAWM, and $p_\omega$ is the state conditional skill prior. The first two terms in Eq.(3.1) correspond to the log-likelihood of demonstrator actions and long-term state-transitions, respectively. The final term represents the KL divergence between our skill posterior and prior, and encourages learning a compressed representation of skills.

The problem with this naive-VI formulation is that the information required to reconstruct the state $s_T$ by the TAWM can be stored in the latent $z$. During planning time, since our reward is a function of states, we can easily exploit the world model by picking latents which have the information of the high reward state, without the corresponding action sequence that will actually get us to that state. In an MDP, the transition dynamics $p(s_{t+1}|s_t, a_t)$ is a part of the environment and not in the control of the agent. With this knowledge, we can derive the form of the true posterior $p(z|\vec{s}, \vec{a})$ of the latent variable model:

$$p(z|\vec{s}, \vec{a}) = \frac{p(s_{1:T}, a_{0,T-1}|z, s_0)p_\omega(z|s_0)}{p(s_{1:T}, a_{0,T-1}|s_0)} \tag{3.2}$$

$$= \frac{(\prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t, z))p_\omega(z|s_0)}{p(s_{1:T}, a_{0:T-1}|s_0)} \tag{3.3}$$

The transition dynamics $p(s_{t+1}|s_t, a_t)$ and $p(s_{1:T}, a_{0:T-1}|s_0)$ do not depend on the skill $z$, and so can be absorbed into a normalization constant $\eta$. This leaves us with the true posterior:

$$p(z|\vec{s}, \vec{a}) = \frac{1}{\eta}((\prod_{t=0}^{T-1} \pi_\theta(a_t|s_t, z))p_\omega(z|s_0) \tag{3.4}$$

Thus, now we need to minimize the KL between our approximate posterior $q_\phi$ and this true posterior. We can now optimize the parameters of our models with an EM style algorithm:

**E-Step:** $\phi$ is updated with a gradient descent step so as to minimize the expected

KL divergence between $q_\phi$ and the true posterior, which is equivalent to minimizing $\mathbb{E}_{\tau\sim\mathcal{D}}\left[\mathbb{E}_{z\sim q_\phi}\left[\log \frac{q_\phi(z|\vec{s},\vec{a})}{\pi_\theta(\vec{a}|\vec{s},z)p_\omega(z|s_0)}\right]\right]$.

**M-Step:** $\theta$, $\psi$, and $\omega$ are updated with a gradient ascent step to maximize the ELBO from Eq. 3.1.

Observing the KL objective to minimize in the E-Step reveals that it is simply optimizing $\phi$ by maximizing the same ELBO, but just without the TAWM reconstruction error term $\mathbb{E}_{\tau\sim\mathcal{D}}\Big[\mathbb{E}_{q_\phi(z|\tau)}\left[\log p_\psi(s_T|s_0,z)\right]\Big]$. Hence, we can replace the EM updates with a single forward-backwards pass for every paramater where we detach the latent $z$ before passing it into the TAWM, thus not backpropagating gradients from it to the encoder.

### 3.3.2 Online Planning with Skill Latents

Once a world model has been learned and a reward function has been specified for a given task, our agent can plan a sequence of skills to maximize its predicted cumulative reward. We use a cross entropy method (CEM) planner to optimize the skill sequence. However, rather than directly optimizing a sequence of $z$ vectors, we plan in a "whitened" version of $\mathcal{Z}$-space. Specifically, we optimize a sequence of $\epsilon$ vectors, where $i$th element of the plan $\epsilon_i \in \mathcal{E}$ is related to $z_i$ according to a shifting and scaling by the learned skill prior mean $\mu_0(s_i)$ and standard deviation $\sigma_0(s_i)$, *i.e.*, $z_i = \mu_0(s_i) + \sigma_0(s_i) \cdot \epsilon_i$.

Planning in $\mathcal{E}$-space allows the agent to search in the space of *normalized biases* away from the demonstration policy. This provides the agent with a well-conditioned search space, where reasonable plan values lie within a (roughly-)unit ball around the origin, regardless of the state, rather than having different state-dependent means and scales. Additionally, planning in $\mathcal{E}$-space allows us to easily warm-start the planning procedure, since our initial plan can easily be sampled from the demonstrator policy.

## 3.4 VQSkills

To the limited expressiveness of the simple conditional Gaussian prior used earlier, we next propose using a VQ-VAE to a model for discrete skills. We also show empirically that it helps reduce exploitation of the world model during planning. We begin by describing the architecture and training process. An interesting quirk with vector quantization is the fact that if the posterior is regularized to roughly be within a unit ball through L2 regularization, we can actually use continuous optimization methods like CEM to optimize a sequence of discrete latent vectors extremely efficiently, and so we next explain this planning process.

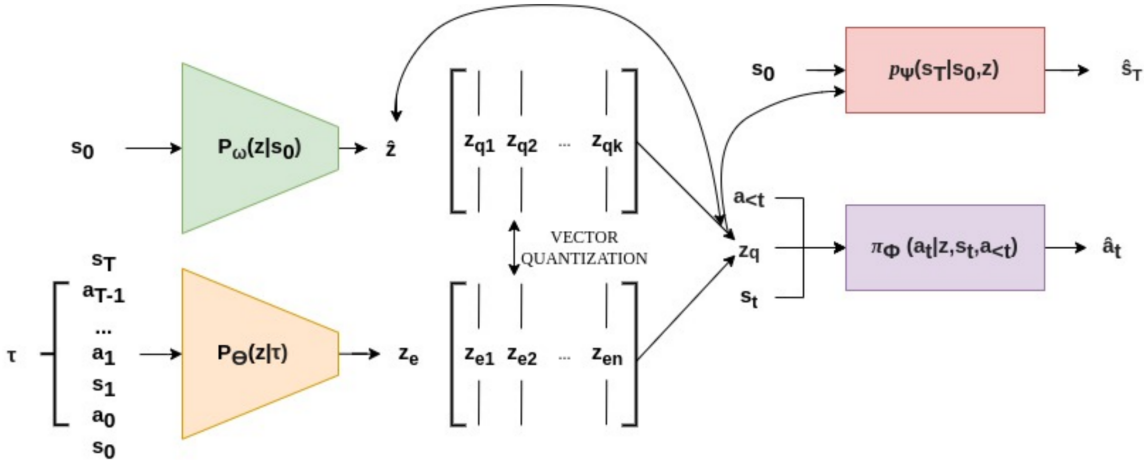



Figure 3.2: **VQSkills Overview** VQSkill consists of a deterministic encoder, a prior, a codebook, a low level policy and a temporally abstract world model. At test time, we do not use the encoder. The prior gives a proposed latent $\hat{z} \sim p_\omega(z|s_0)$, which we center our CEM optimizer on. In situations where behavioral cloning suitable(for example in the goal conditioned antmaze setting), the prior is used as a high level policy.

### 3.4.1 VQSkill Model

VQSkill extends the skill model described earlier to include vector quantization and a temporally abstract world model. We have an encoder $p_\theta(z|\tau)$ which outputs n vectors of size $z_{dim}$ which we label $z_{e1}, z_{e2}, ..., z_{en}$. The codebook $Q$ consists of $k$ vectors of size $z_{dim}$. We include a prior model $p_\omega(z|s_0)$. Finally, we have 2 decoders- the action

**Algorithm 2** VQSkill training

Randomly initialize parameters in $p_\theta, p_\omega, \pi_\phi, p_\psi, Q$
**while** models not converged **do** $\tau_{1,...,m} \sim \mathcal{D}$ ▷ Sample m trajectories from dataset
$L := 0$
**for** $i = 1, 2, \ldots, m$ **do** $z_{e1}, z_{e2}, ..., z_{en} = p_\theta(\tau_i)$ ▷ Get n latents from deterministic encoder
$z_{q1}, ..., z_{qn} = \text{vector\_quantization}(Q, z_{e1}, ..., z_{en})$▷ Nearest neighbor column in Q for $z_{ei}$
$z_q = \text{concat}(z_{q1}, ..., z_{qn})$, $z_e = \text{concat}(z_{e1}, ..., z_{en})$
$L+ = \frac{1}{m}\alpha||sg(z_e) - z_q||^2 + \beta||z_e - sg(z_q)||^2 + \gamma||z_e||^2$ ▷ Embedding Loss
$z_q := z_e + sg(z_q - z_e)$ ▷ Trick to propagate gradients
**for** $t = 0, 1, \ldots, T-1$ **do** $L+ = -\frac{1}{mT}\log \pi_\phi(a_t|z_q, s_t, a_{<t})$▷ Action decoder log prob for loss $L+ = -\frac{1}{m}\log p_\psi(s_T|z, s_0)$ ▷ World model log prob for loss
$L+ = -\frac{1}{m}\log p_\omega(z_q|s_0)$ ▷ Prior latent log prob of quantized vector Use Adam or other optimizer to minimize $L$

decoder/low level policy $\pi_\phi(a_t|z, s_t, a_{<t})$ and the world model which predicts the state $T$ timesteps in the future $p_\psi(s_T|z, s_0)$. The training pipeline is given in algorithm 2, and shown in figure 3.2 above.

The encoder is a deterministic network which takes an input a trajectory and outputs a sequence of latent vectors, which are vector quantized to vectors in the codebook. All other networks are parameterized as gaussians, outputting a mean and standard deviation. The prior tries to learn a distribution over the possible vector quantized latents. It might be expected that at convergence the prior should predict a point mass (standard deviation $\sigma$ near 0) for a given trajectory input since the latents are fixed vectors and the encoder is deterministic. However, in situations where there can be multiple behavioral modes such as near an intersection in antmaze, there are multiple possible skills that can be executed, depending on the goal for that episode in the dataset which the encoder is unaware of(we treat all demonstrations as undirected). In these cases, the prior tries to spread out probability mass over the possible latents. We regularize the output of the encoder to remain roughly in a unit ball around the origin, thus packing the skill latents relatively close together. We leverage this to do continuous planning over discrete skills at test time, where we can navigate to any goal waypoint in the maze.

### 3.4.2 Online Planning with Skill Latents

While discrete spaces can be searched using techniques like Monte-Carlo Tree Search, we instead opt to do continuous optimization. By doing this, we can make use of the intuition that even though they are discrete vectors, due to the mechanism of vector quantized training and regularization of the latents, we end up with a latent space where skill latents that are closer to each other in euclidean space correspond to similar behaviors as compared to a skills that are mutually seperated by a longer distance. Thus, we can almost treat the discrete space as a continuous latent space, but while maintaining many of the advantages of discrete latents over continuous latents.
In the equations below, $\mu_\omega(s)$ and $\sigma_\omega(s)$ are the mean and variance of the prior $p_\omega(z|s)$. We use CEM to plan over the noise vectors $\epsilon_i$ to optimize the objective below:

$$\begin{aligned} &\max_{\epsilon_{1,\ldots,H}} \sum_{i=1}^{H} R(s_{T\times i}) \\ s.t. \quad & s_{T\times i} = \mathbb{E}[p_\psi(s_T|z_i, s_{T\times(i-1)})], \\ & z_i^{(0)} = \mu_\omega(s_{T\times(i-1)}) + \sigma_\omega(s_{T\times(i-1)})\epsilon_i \\ & z_i = \text{vector\_quantization}(Q, z_i^{(0)}) \end{aligned} \tag{3.5}$$

This formulation assumes the reward to optimize can be a function of a $H$ states each seperated by $T$ timesteps. In tasks where we wish to navigate to some goal this can be acceptable. Planning at this lower frequency can allow us to look forward much further than we reasonably could planning at the smallest time scale.

We optimize the objective using Cross-Entropy Method(CEM), which is an efficient zeroth order optimization algorithm. Every iteration, we select a top fraction of elite candidates with the highest reward per batch, and use MLE to fit a new distribution (gaussian here) to these candidates which we sample from during the next iteration. Each iteration, the entire candidate batch can be evaluated in parallel. In This version of CEM for discrete planning, after we generate each new $z_i^{(0)}$ using $\epsilon_i$, we quantize it to the closest vector in the codebook.

The choice to plan over $\epsilon_i$ which is scaled by the standard deviation of the prior

latent distribution allows us to plan accounting for the actual necessity for planning at a given state. We initialize $\epsilon_i \sim \mathcal{N}(0, 1)$. This corresponds to our initialization of $z_i$ being sampled from $p_\omega$. Small standard deviation indicates that the prior mean itself is likely close to the optimal skill, so the samples are drawn close to it, which means only skills close to the mean are considered. In states where planning is important, we expect to see a higher standard deviation in the prior distribution, which encourages exploration.

## 3.5 Experimental Evaluation and Analysis

We conduct experiments on 4 benchmarks - 1)Maze2D, 2)Antmaze and 3)FrankaKitchen from D4RL, and 3)Carla NoCrash. Maze2d is a maze navigation environment which requires stitching together trajectories in the dataset in order to learn to navigate towards any point on the maze. Antmaze is a much more difficult version of the maze challenge where apart from planning the path through the maze, the agent must also learn to control an ant robot with moderately complicated dynamics. FrankaKitchen involves controlling a Franka arm to do specified tasks in a kitchen setting such as moving a kettle, opening a microwave etc. Carla NoCrash is an autonomous driving benchmark in the Carla simulator, where the agent must control a car to drive through busy urban roads while adhering to traffic rules.
In these tasks, we compare against popular offline RL algorithms and behavioral cloning. We include Decision Diffuser (Ajay et al. [3]) since it is a SOTA offline RL algorithm which excels at skill stitching using diffusion, which we investigate further in the next chapter.

### 3.5.1 Maze2D

We test on two variants of the maze2D task: medium and large. In both variants, a point-mass agent moves around a 2-dimensional maze. The goal of the task is for the agent to navigate from one corner of the maze to the opposite corner. The size of the maze differs between the two environments, with large using the larger of the two. In both cases, our agent was trained on the corresponding task dataset provided by D4RL, which contain a single long state-action sequence of the agent moving about the maze, towards randomly-chosen waypoints. An episode is considered a success if the agent comes within a 0.5-unit radius of the goal. The cost assigned to a predicted trajectory during planning is the negative $\mathcal{L}_2$ distance between the goal and the nearest predicted state to the goal. Rewards are therefore sparse in the sense that an entire trajectory is assigned one scalar reward value; no per-timestep rewards are used.

Both OPOSM and VQSkills achieve **100%** success rate on both Maze2D tasks.

For VQSkills, we use a skill model with $z_{dim} = 8$,$n = 4$(number of discrete latents per skill),$k = 8$(number of latents in codebook). We set $T = 20$, where $T$ is the number of time steps ahead the world model $p_\psi(s_T|z, s_0)$ predicts. We choose same $T$ for OPOSM, and $z_{dim} = 64$. This environment served as a sanity test before the significantly increased challenge of antmaze.

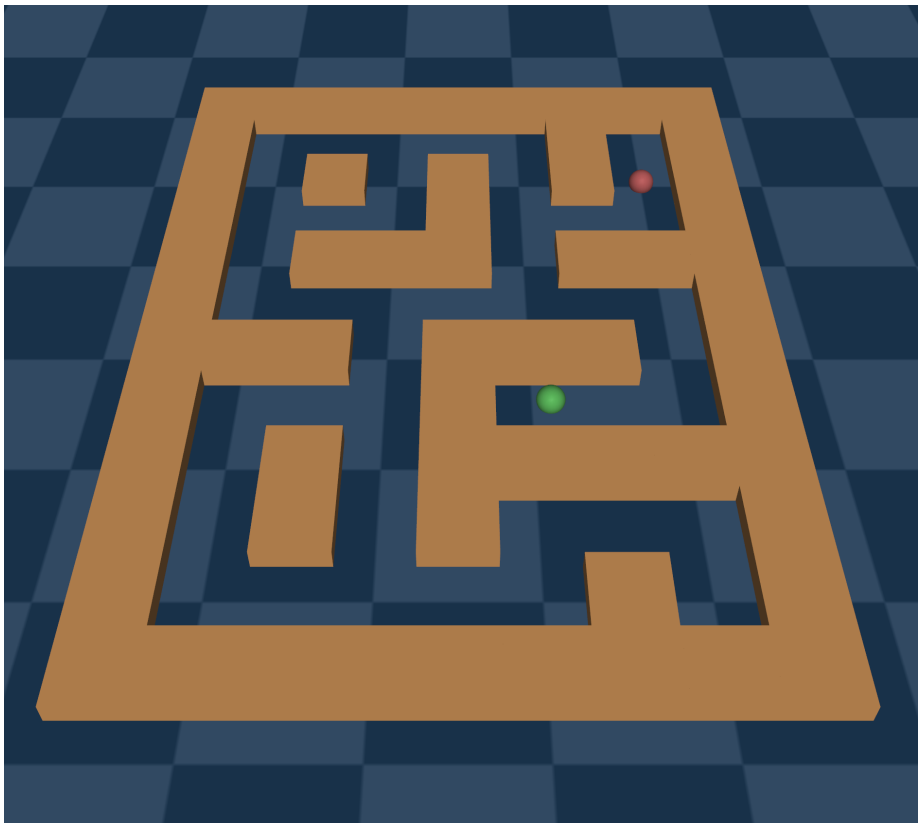


Figure 3.3: Maze2d environment consists of a simple pointmass agent navigating to a given goal location in a maze.

### 3.5.2 AntMaze

We test on the antmaze-medium and antmaze-hard. Using MPC allows us to learn from undirected examples and do online planning to reach goals in the test setup. We get state of the art results on these tasks

We plan with $H = 5$ (number of world model calls per rollout), and $T = 20$, which means our effective planning horizon is 100 timesteps. The cost function(negative reward) that we use for both antmaze and maze2d is $C(s_T, s_{2T}, ..., s_{HT}) = min_i||s_{iT}[:2] - goal||^2$ which is the minimum distance(in xy coordinates) to the goal reached by the states in the rollout.

VQSkills performs very similarly to OPOSM in antmaze-medium, and beats it in antmaze-large. The medium setting does not require complex high level path planning due to the simplicity of the layout, which might have made it difficult to improve performance with online planning. Qualitatively, the ant controlled with VQSkills seemed to rarely take a wrong turn and its failures were mostly wall collisions that toppled it. The OPOSM agent on the other hand seemed more prone to taking wrong paths through the maze, despite the predicted cost after CEM optimization being low. This suggests that the continuous skill latents when optimized may exploit the world model. We directly test this hypothesis by greatly increasing the CEM population size from 100 to 10000 and the number of iterations from 10 to 1000 and then evaluating the model on 50 runs. This over-optimization breaks the OPOSM planning and the performance drops from roughly 70% on antmaze-large to around 46%, whereas the VQSkills much more modestly drops from 74% to 64% (results included in table 3.4). We believe this is due to VQSkills constraining the latents to only the discrete vectors in the codebook, which mitigates over-exploitation of the world model. Snapping to the closest discrete latent may also limit the divergence from the prior.

We expect to see a similar drop off happen if we increase the number of latents in VQSkill by a large amount, which would exponentially increase the number of optimization variables bringing it closer to the continuous case. We compare the best skill model with $k = 8$ to others with increasing values of $k$ on antmaze-large, and also overoptimize them. With models that have a large codebook, the overoptimized dropoff is more significant, worse than OPOSM. We also find that reducing $k$ (size of codebook) too much also hurts performance, due to reduced expressivity. The degenerate case when $k = 1$ means the latent has no information, reducing to behavioral cloning. This shows that choosing the number of latent codes is extremely important, and a good balance must be struck. Note that all these experiments used $n = 4$ which means there are 4 latent codes per skill. Increasing this parameter would

also exponentially grow the optimization complexity.

Table 3.1: Comparison of algorithms on antmaze. OPOSM and VQSkill outperform all methods except CQL+OPAL on antmaze-medium.

| Task | **VQSkill** | **OPOSM** | BC | CQL | IQL | DD |
|---|---|---|---|---|---|---|
| antmaze-medium-diverse | $\mathbf{79.67 \pm 4.5}$ | $78.29 \pm 4.32$ | 0.0 | 53.7 | 70.0 | 24.6 |
| antmaze-large-diverse | $\mathbf{74.33 \pm 4.9}$ | $70.2 \pm 4.6$ | 0.0 | 14.9 | 47.5 | 7.5 |

Table 3.2: Comparison of various values of $k$ on antmaze-large-diverse, the number of vectors in codebook $Q$. There is significant drop off in performance with overoptimization when $k$ is large. We run 50 episodes of evaluation for each model and report fraction of successes.

| Task | k=1 | k=4 | **k = 8** | k=32 | k=128 | k=1024 | OPOSM |
|---|---|---|---|---|---|---|---|
| short optimization | 0.0 | 66 | **74.33** | 72 | 68 | 68 | 70.2 |
| overoptimized | – | 48(24/50) | **64(32/50)** | 58(29/50) | 36(18/50) | 18(9/50) | 46(23/50) |

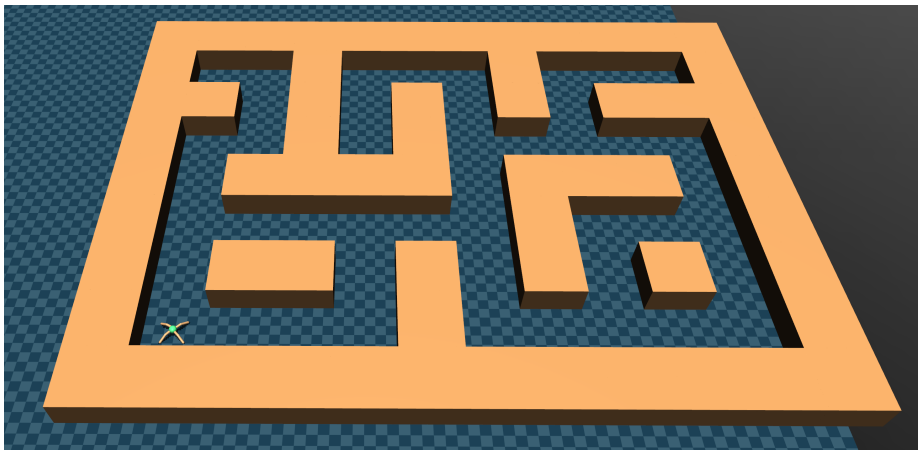


Figure 3.4: AntMaze environment consists of an ant robot agent navigating to a given goal location in a maze. This ant agent has more complex dynamics than the pointmass in Maze2D.

### 3.5.3 FrankaKitchen

We train our skill model on different datasets, referred to as *mixed* and *partial*. The mixed dataset consists of non-task-directed demonstrations, and is typically considered the hardest to learn from, while the partial dataset consists of partially task-directed demonstrations.

We find that OPOSM is outperformed by Decision Diffuser on the kitchen tasks, which we hypothesize is due to the fact that these tasks require constraining to the multimodal behavior support which requires a more expressive prior than the conditional Gaussian used by our VAE. Currently, we do not explicitly constrain plans computed by OPOSM to remain within the support of the dataset. We outperform other algorithms under consideration.

VQSkills performs relatively poorly in the FrankaKitchen tasks, with the training process being very unstable. Vector Quantized networks suffer from training instability due to approximation with straight through gradients (Huh et al. [25]). If not for this instability, we suspect the VQ-VAE could have outperformed the standard VAE in OPOSM, since discrete latent posteriors can be more expressive than a conditional Gaussian. In the next chapter, we propose using Diffusion models which are stable to train while also having better expressiveness than VQ-VAEs. The Latent Diffusion skill model proposed in chapter 4 performs significantly better.

Table 3.3: Comparison of algorithms on kitchen tasks.

| Task | **VQSkill** | **OPOSM** | BC | CQL | IQL | DD |
|---|---|---|---|---|---|---|
| kitchen-mixed-v0 | 47 | 54.5 | 51.5 | 52.4 | 51.0 | **62.3** |
| kitchen-partial-v0 | 54.33 | 65.25 | 38.0 | 50.1 | 46.3 | **67.8** |

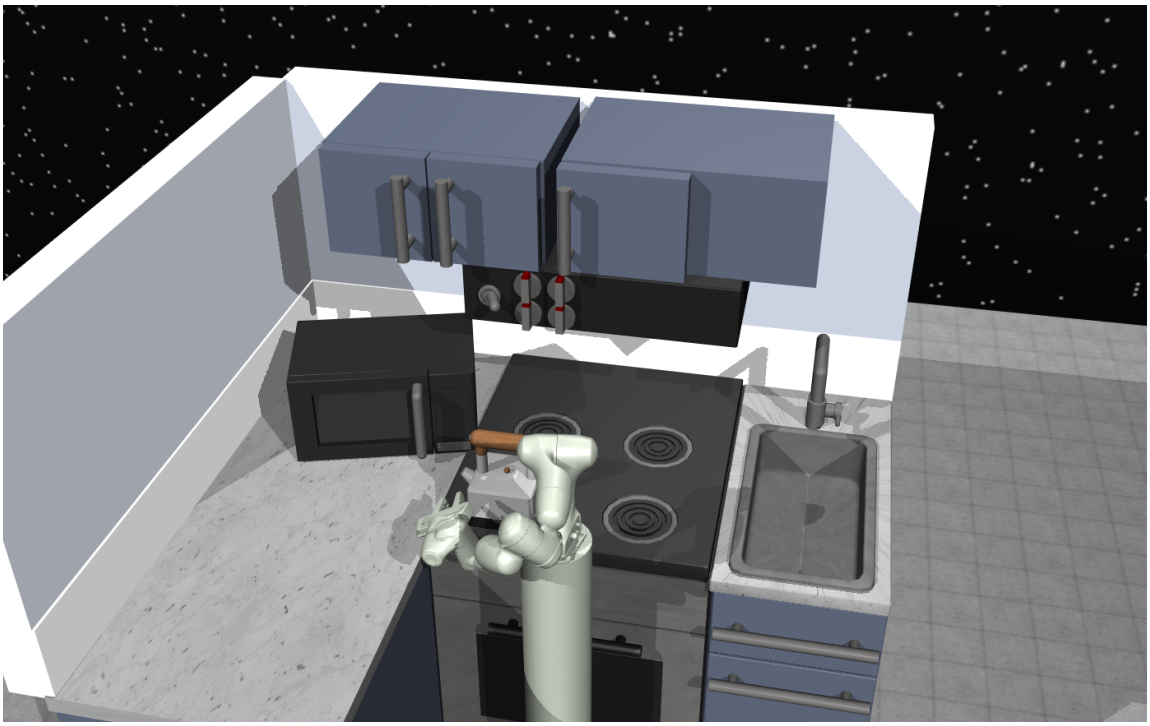


Figure 3.5: In the kitchen tasks, the Franka arm must be controlled to perform several tasks in the kitchen environment. Only certain tasks provide rewards, so the agent must learn to only do those in the required sequence order.

### 3.5.4 Carla NoCrash

We use the modified low dimensional observation space used in SPLT (Villaflor et al. [70]) and MPPO (Huang [24]) which consists of the current steering angle and speed of the car, the next intermediate waypoint heading, the distance and velocity of the nearest car in front, distance and color of the nearest traffic light. The NoCrash environment requires the agent to do lane following, avoid collisions and traffic light violations, and navigate to the goal. Due to simple dynamics of the car to be controlled, lane following is generally a simple task, and the difficulty arises from knowing how to avoid collisions with other agents. For online planning, we use a simple reward function of the state $R(s) = (vehicle\_speed)^2 - traffic\_violation\_penalty$.
where $traffic\_violation\_penalty = 200$.

The datasets are collected on Carla Town1 by an expert autopilot with added noise, and the evaluations are done on Carla Town2, where there are 25 routes. We run evaluation on NoCrash Dense, which is the most challenging heavy traffic setting.

Due to the relative simplicity in modes of behavior in NoCrash, we train a VQSkill model where the skill consists of a single latent($n = 1$), and the codebook consists of $k = 4$ total latents. This means there are only 4 possible skills. After training, we observe that these 4 skills collapse to just 2 final skills. One of these skills if active causes the car to stop regardless of situation, while the other causes the car to move forward along the required trajectory, taking turns and following the lane perfectly. The prior learns to predict which of these 2 skills to run at every step, and it essentially learns when to stop and when to move. Simply running the prior as a higher level policy without any online planning actually gets better performance than naive behavioral cloning. We do not use CEM for online optimization here, instead simply trying out all combinations of skills. We run with $H = 4$ with $T = 10$, which is 40 timesteps lookahead planning, which due to the simulator tick time of 10 Hz is 4 seconds lookahead. Tring out all skills involves $4^4 = 256$ forward passes of the world model, which can be done in a single batch. With online planning, we beat other online and offline RL algorithms that have been tried in the modified NoCrash Dense benchmark. OPOSM however does not efficiently learn this policy, performing on par with behavioral cloning when running the prior, and worse than comparing algorithms with planning. Since exhaustive search is not an option, we

must use CEM for optimizing OPOSM, and we also find that optimizing any more than just 5 steps of CEM completely fails every run due to exploitation of the world model. To compare this, we also train a VQSKills model with $n = 4$ and $k = 8$. The performance is worse than the $n = 1, k = 4$ case, but better than OPOSM, and overoptimization does not make performance any worse.

Table 3.4: Comparison of algorithms for NoCrash Dense Town2 routes. SPLT and Behavioral Cloning are offline, while MPSAC and MPPO are online.

| | **VQSkill($\mathbf{n = 1}$)** | VQSkill(n=4) | OPOSM | SPLT | MPPO | MPSAC | BC |
|---|---|---|---|---|---|---|---|
| Town2 success (%) | **97.6** | 90.4 | 87.2 | 96.3 | 96 | 92 | 84 |

## 3.6 Limitations

### OPOSM

1. Standard VAEs are not as expressive as more complex generative models, and as such may sample points of low likelihood form multimodal distributions due to mode averaging. This is investigated further in the next chapter.
2. The assumption of a reward function that depends only on states in which skills begin or terminate may not hold for many tasks. This assumption is made because these are the only states predicted by our TAWM.
3. The use of fixed-length skills is limiting because for many tasks, the most natural decomposition may result in subtasks (and thus skills) that require differing amounts of time to be completed.

### VQSkills

1. The same limitations regarding fixed skill length and state conditioned reward function apply.
2. While a discrete skill model is more expressive than the VAE in OPOSM, it suffers from an unstable training process. The learning algorithm is also extremely sensitive to hyperparameters, such as the regularizing constant $\gamma$ that limits the norm of the encoder output latent $z_e$. The training process must

be carefully monitored to make sure the skills do not collapse, at which point training must be restarted.

# Chapter 4

# Latent Diffusion for Offline RL

## 4.1 Introduction

Framing offline RL as a generative modeling problem has gained significant traction (Chen et al. [5], Janner et al. [28]); however, the performance is dependent on the power of the generative models used. These methods either avoid learning a Q-function or rely on other offline Q-learning methods. Recently diffusion models (Sohl-Dickstein et al. [60], Song and Ermon [62]), have emerged as state-of-the-art generative models for conditional image-generation (Ramesh et al. [47]). Rather than avoiding Q-learning, we model the behavioral policy with diffusion and use this to avoid extrapolation error through batch-constraining.

Previous diffusion-based sequence modeling methods in offline RL diffused over the raw state-action space. However, the low-level trajectory space tends to be poorly suited for reasoning. Prior works (Pertsch et al. [45], Ajay et al. [2]) have proposed to instead reason in more well-conditioned spaces composed of higher-level behavioral primitives, and we demonstrated this as well in the previous chapter. Such temporal abstraction has been shown to result in faster and more reliable credit assignment (Machado et al. [40], Mann and Mannor [41]), particularly in long-horizon sparse-reward tasks. **We harness the expressivity of powerful diffusion generative models to reason with temporal abstraction and improve credit assignment.**

Inspired by the recent successes of Latent Diffusion Models (LDMs) (Rombach et al. [49], Jun and Nichol [31]), we propose learning similar latent trajectory representations

for offline RL tasks by encoding rich high-level behaviors and learning a policy decoder to roll out low-level action sequences conditioned on these behaviors. The idea is to diffuse over semantically rich latent representations while relying on powerful decoders for high-frequency details. Prior works which explored diffusion for offline RL (Janner et al. [29], Ajay et al. [3]) directly diffused over the raw state-action space, and their architectural considerations for effective diffusion models limited the networks to be simple U-Nets (Ronneberger et al. [50]). The separation of the diffusion model from the low-level policy allows us to model the low-level policy using a powerful autoregressive decoder.

We perform state-conditioned latent diffusion on the learnt latent space and then learn a Q-function over states and corresponding latents. During Q-learning, we batch-constrain the candidate latents for the target Q-function using our expressive diffusion prior, thus mitigating extrapolation error. Our final policy samples latent skills from the LDM, scores the latents using the Q-function and executes the best behavior with the policy decoder. We refer to our method as Latent Diffusion-Constrained Q-learning (LDCQ). Our method significantly improves results over previous offline RL methods, which suffer from extrapolation error or have difficulty in credit assignment in long-horizon sparse reward tasks.

## 4.2 Related Work

### 4.2.1 Offline RL

As discussed previously, offline RL poses the challenge of distributional shift while stitching suboptimal trajectories together. Conservative Q-Learning (CQL) (Kumar et al. [36]) tries to constrain the policy to the behavioral support by learning a pessimistic Q-function that lower-bounds the optimal value function. Implicit Q-Learning (IQL) (Kostrikov et al. [34]) tries to avoid extrapolation error by performing a trade-off between SARSA and DQN using expectile regression. However, it achieves the optimal batch-constrained policy only as their expectile parameter $\tau \to 1$, which leads to an increasingly difficult-to-optimize objective. Our method instead learns the optimal batch-constrained Q-function without introducing any pessimism or trade-off.

Inspired by notable achievements of generative models in various domains including text-generation (Vaswani et al. [69]), speech synthesis ([33]) and image-generation (Ramesh et al. [47]), Chen et al. [5] proposed to use a generative model for offline RL and bypass the need for Q-learning or bootstrapping altogether with *return-conditioning* (Kumar et al. [35], Srivastava et al. [64]). While these ideas have found success, getting a good return estimate for arbitrary states is not trivial and conditioning on returns outside the support of the training dataset can lead to the generative model producing low-value out-of-distribution sequences. Our method instead avoids return-conditioning and formulates a solution with batch-constraining which uses generative models to model the data distribution and use it to generate candidate actions to learn a Q-function without extrapolation-error (Fujimoto et al. [13]). This formulation relies on the assumption that sampling from the generative model does not sample out-of-support samples, which has been difficult to achieve with previously used generative models in offline RL.

Our method circumvents this problem with the latent diffusion model. Further, to effectively address the problem of long horizon stitching, Pertsch et al. [45] and Ajay et al. [2] proposed learning policies in latent-trajectory spaces. However, they have to rely on a highly constrained latent space which is not rich enough for the downstream policy. This is due to the limitations of the generative model used like VAEs. Our proposed method to use latent diffusion, which can model complex distributions,

allows for the needed flexibility in the latent space for effective Q-learning and the final policy.

### 4.2.2 Diffusion Probabilistic Models

Recently, diffusion models (Sohl-Dickstein et al. [60], Song and Ermon [62]) have emerged as state-of-the-art generative models for conditional image-generation (Ramesh et al. [47], Saharia et al. [52]), super-resolution (Saharia et al. [51]) and inpainting (Lugmayr et al. [39]). They are a much more powerful class of generative model compared to Variational Autoencoders (VAEs), and benefit from a more stable training process as compared to Generative Adversarial Networks (GANs) ([15]). Recent works in offline RL (Janner et al. [29], Ajay et al. [3]) have proposed using diffusion to model trajectories and showcased its effectiveness in stitching together behaviors to improve upon suboptimal demonstrations. However, Janner et al. [29] make the assumption that the value function is learnt using other offline Q-learning methods and their classifier-guided diffusion requires querying the value function on noisy samples, which can lead to extrapolation-error. Similarly, Ajay et al. [3] can suffer from distributional shift, as it relies on return-conditioning, and maximum returns from arbitrary states can be unknown without having access to a value function. Our work proposes a method for learning Q-functions in latent trajectory space with latent diffusion while avoiding extrapolation-error and facilitating long horizon trajectory stitching and credit assignment.

## 4.3 Latent Diffusion Reinforcement Learning

We begin by describing the two-stage training process for obtaining the low-level policy and the high-level latent diffusion prior. Next, we discuss how to use this prior to train a temporally abstract Q-function while avoiding bootstrapping error, and then use this Q-function during the policy execution phase. We finally describe an additional method to use the latent diffusion prior with goal-conditioning, which is more suitable for certain navigation tasks. Model architectures and hyperparameter choices are detailed in the appendix.

### 4.3.1 Two-Stage LDM training

#### Latent Representation and Low-Level Policy

The first stage in training the latent diffusion model is comprised of learning a latent trajectory representation. This means, given a dataset $\mathcal{D}$ of $H$-length trajectories $\boldsymbol{\tau}_H$ represented as sequences of states and actions, $\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \cdots \mathbf{s}_{H-1}, \mathbf{a}_{H-1}$, we want to learn a low-level policy $\pi_\theta(\mathbf{a}|\mathbf{s}, \mathbf{z})$ such that $\mathbf{z}$ represents high-level behaviors in the trajectory. This is done using a $\beta$-Variational Autoencoder (VAE). Specifically, we maximize the evidence lower bound (ELBO):

$$\mathcal{L}(\theta, \phi, \omega) = \mathbb{E}_{\boldsymbol{\tau}_H \sim D}[\mathbb{E}_{q_\phi(\mathbf{z}|\boldsymbol{\tau}_H)}[\sum_{t=0}^{H-1} \log \pi_\theta(\mathbf{a}_t \,|\, \mathbf{s}_t, \mathbf{z})] - \beta D_{KL}(q_\phi(\mathbf{z} \,|\, \boldsymbol{\tau}_H) \;||\; p_\omega(\mathbf{z} \,|\, \mathbf{s}_0))] \tag{4.1}$$

where $q_\phi$ represents our approximate posterior over $\mathbf{z}$ given $\boldsymbol{\tau}_H$, and $p_\omega$ represents our conditional Gaussian prior over $\mathbf{z}$, given $\mathbf{s}_0$. Note that unlike BCQ, which uses a VAE's conditional Gaussian prior as the state-conditioned generative model, our latent diffusion model only uses the $\beta$-VAE to learn a latent space to diffuse over. As such, the prior $p_\omega$ is simply a loose regularization of this latent space, and not a strong constraint. This is facilitated by the ability of latent diffusion models to later sample from such complex latent distributions. As discussed in the earlier chapters, prior works (Pertsch et al. [45], Ajay et al. [2]) have learned latent space representations of skills using VAEs. Their use of weaker Gaussian priors forces them to use higher values of the KL penalty multiplier $\beta$ to ensure the latents are well

regularized. However, doing so restricts the information capacity of the latent, which limits the variation in behaviors captured by the latents. As we show in section 4.4.1, increasing the horizon $H$ reveals a clear separation of useful behavioral modes when the latents are weakly constrained.

The low-level policy $\pi_\theta$ is represented as an autoregressive model which can capture the fine details across the action dimensions, and is similar to the decoders used by Ghasemipour et al. [14] and Ajay et al. [2]. While all the environments we test in this work use continuous action spaces, the use of latent diffusion allows the method to easily translate to discrete action spaces too, since the decoder can simply be altered to output a categorical distribution while the diffusion process remains unchanged.

### Latent Diffusion Prior

For training the diffusion model, we collect a dataset of state-latent pairs ($\mathbf{s}_0$, $\mathbf{z}$) such that $\boldsymbol{\tau}_H \sim \mathcal{D}$ is a $H$-length trajectory snippet, $\mathbf{z} \sim q_\phi(\mathbf{z} \,|\, \boldsymbol{\tau}_H)$ where $q_\phi$ is the VAE encoder trained earlier, and $\mathbf{s}_0$ is the first state in $\boldsymbol{\tau}_H$. We want to model the prior $p(\mathbf{z} \,|\, \mathbf{s}_0)$, which is the distribution of the learnt latent space $\mathbf{z}$ conditioned on a state $\mathbf{s}_0$. This effectively represents the different behaviors possible from the state $\mathbf{s}_0$ as supported by the behavioral policy that collected the dataset. To this end, we learn a conditional latent diffusion model $p_\psi(\mathbf{z} \,|\, \mathbf{s}_0)$ by learning the time-dependent denoising function $\mu_\psi(\mathbf{z}_t, \mathbf{s}_0, t)$, which takes as input the current diffusion latent estimate $\mathbf{z}_t$ and the diffusion timestep $t$ to predict the original latent $\mathbf{z}_0$. Like Ramesh et al. [47] and Jun and Nichol [31], we found predicting the original latent $\mathbf{z}_0$ works better than predicting the noise $\boldsymbol{\epsilon}$. We reweigh the objective based on the noise level according to Min-SNR-$\gamma$ strategy (Hang et al. [19]). This re-balances the objective, which otherwise is dominated by the loss terms corresponding to diffusion time steps closer to $T$. Concretely, we modify the objective in Eq. 2.14 to minimize:

$$\mathcal{L}(\psi) = \mathbb{E}_{t\sim[1,T],\boldsymbol{\tau}_H\sim\mathcal{D},\mathbf{z}_0\sim q_\phi(\mathbf{z}|\boldsymbol{\tau}_H),\mathbf{z}_t\sim q(\mathbf{z}_t|\mathbf{z}_0)}[\min\{\mathrm{SNR}(t),\gamma\}(||\ \mathbf{z}_0 - \mu_\psi(\mathbf{z}_t, \mathbf{s}_0, t)\ ||^2)] \tag{4.2}$$

Note that $q_\phi(\mathbf{z} \mid \boldsymbol{\tau}_H)$ is different from $q(\mathbf{z}_t \mid \mathbf{z}_0)$, the former being the approximate posterior of the trained VAE, while the latter is the forward Gaussian diffusion noising process. We use DDPM (Ho et al. [23]) to sample from the diffusion prior in this work due to its simple implementation.

As proposed in Ho and Salimans [21], we use classifier-free guidance during the sampling process. This modifies the original training setup to learn both a conditional $\mu_\psi(\mathbf{z}_t, \mathbf{s}_0, t)$ and an unconditional model. The unconditional version, is represented as $\mu_\psi(\mathbf{z}_t, \varnothing, t)$ where a dummy token $\varnothing$ takes the place of $\mathbf{s}_0$. The following update is then used to generate samples: $\mu_\psi(\mathbf{z}_t, \varnothing, t) + w(\mu_\psi(\mathbf{z}_t, \mathbf{s}_0, t) - \mu_\psi(\mathbf{z}_t, \varnothing, t))$, where $w$ is a tunable hyper-parameter. Increasing $w$ results in reduced sample diversity, in favor of samples with high conditional density.

### 4.3.2 Latent Diffusion-Constrained Q-Learning (LDCQ)

As described in chapter 2, in the batch-constrained, we can mitigate extrapolation error by choosing a policy that only selects actions with support in the dataset batch. With function approximation, we can learn a generative model of the behavior policy and constrain our candidate actions with this model:

$$\pi(\mathbf{s}) = \underset{\mathbf{a}_i \sim \pi_\psi(\mathbf{a}|\mathbf{s})}{\operatorname{argmax}} Q(\mathbf{s}, \mathbf{a}_i) \tag{4.3}$$

However, in many offline RL datasets, the behavior policy is highly multimodal either due to the demonstrations being undirected, or because the behavior policy is actually a mixture of unimodal policies, making it difficult for previously used generative models like VAEs to sample from the distribution accurately. The multimodality of this policy is further exacerbated with increases in temporal abstraction in the latent space, as we show in section 4.4.1. We propose using latent diffusion to model this distribution, as diffusion is well suited for modelling such multi-modal distributions. We propose to learn a Q-function in the latent action space with latents sampled from the diffusion model. Specifically, we learn a Q-function $Q(\mathbf{s}, \mathbf{z})$, which represents the action-value of a latent action sequence $\mathbf{z}$ given state $\mathbf{s}$. At test time, we generate candidate latents from the diffusion prior $p_\psi(\mathbf{z} \,|\, \mathbf{s})$ and select the one which maximizes the learnt Q-function. We then use this latent with the low-level policy $\pi_\theta(\mathbf{a}_i \,|\, \mathbf{s}_i, \mathbf{z})$ to generate the action sequence for $H$ timesteps.

**Algorithm 3** Latent Diffusion-Constrained Q-Learning (LDCQ)

**Input:** prioritized-replay-buffer $\mathcal{B}$, horizon $H$, target network update-rate $\rho$, mini-batch size $N$, number of sampled latents $n$, maximum iterations $M$, discount-factor $\gamma$, latent diffusion denoising function $\mu_\psi$, variance schedule $\alpha_1, \ldots, \alpha_T$, $\bar{\alpha}_1, \ldots, \bar{\alpha}_T$, $\beta_1, \ldots, \beta_T$.
Initialize Q-networks $Q_{\Theta_1}$ and $Q_{\Theta_2}$ with random parameters $Q_{\Theta_1}$, $Q_{\Theta_2}$ and target Q-networks $Q_{\Theta_1^{target}}$ and $Q_{\Theta_2^{target}}$ with $\Theta_1^{target} \leftarrow \Theta_1$, $\Theta_2^{target} \leftarrow \Theta_2$
**for** $iter = 1$ to $M$ **do**
    Sample a minibatch of $N$ transitions $\{(\mathbf{s}_t, \mathbf{z}, r_{t:t+H}, \mathbf{s}_{t+H})\}$ from $\mathcal{B}$
    Sample $n$ latents for each transition: $\mathbf{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    **for** $t = T$ to 1 **do** ▷ DDPM Sampling
        $\hat{\mathbf{z}} = \mu_\psi(\mathbf{z}_t, \varnothing, t) + w(\mu_\psi(\mathbf{z}_t, \mathbf{s}_{t+H}, t) - \mu_\psi(\mathbf{z}_t, \varnothing, t))$
        $\mathbf{z}_{t-1} \sim \mathcal{N}(\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{z}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\hat{\mathbf{z}}, \mathbb{I}(t > 1)\beta_t \mathbf{I})$
    **end for**
    Compute the target values $y = r_{t:t+H} + \gamma^H \{\max_{\mathbf{z}_0}\{\min_{j=1,2} Q_{\Theta_j^{target}}(\mathbf{s}_{t+H}, \mathbf{z}_0)\}\}$
    Update $Q$-networks by minimizing the loss: $\frac{1}{N}||y - Q_\Theta(\mathbf{s}_t, \mathbf{z})||_2^2$
    Update target $Q$-networks: $\Theta^{target} \leftarrow \rho\Theta + (1-\rho)\Theta^{target}$
**end for**=0

### Training

We collect a replay buffer $\mathcal{B}$ for the dataset $\mathcal{D}$ of $H$-length trajectories and store transition tuples $(\mathbf{s}_t, \mathbf{z}, r_{t:t+H}, \mathbf{s}_{t+H})$ from $\boldsymbol{\tau}_H \sim \mathcal{D}$, where $\mathbf{s}_t$ is the first state in $\boldsymbol{\tau}_H$, $\mathbf{z} \sim q_\phi(\mathbf{z} \mid \boldsymbol{\tau}_H)$ is the latent sampled from the VAE approximate posterior, $r_{t:t+H}$ represents the $\gamma$-discounted sum of rewards accumulated over the $H$ time-steps in $\boldsymbol{\tau}_H$, and $\mathbf{s}_{t+H}$ represents the state at the end of $H$-length trajectory snippet. The Q-function is learned with temporal-difference updates (Sutton and Barto [66]), where we sample a batch of latents for the target argmax using the diffusion prior $p_\psi(\mathbf{z} \mid \mathbf{s}_{t+H})$. This should only sample latents which are under the support of the behavioral policy, and hence with the right temporal abstraction, allows for stitching skills to learn an optimal policy grounded on the data support. The resulting Q update can be summarized as:

$$Q(\mathbf{s}_t, \mathbf{z}) \leftarrow (r_{t:t+H} + \gamma^H Q(\mathbf{s}_{t+H}, \underset{\mathbf{z}_i \sim p_\psi(\mathbf{z}|\mathbf{s}_{t+H})}{\operatorname{argmax}} (Q(\mathbf{s}_{t+H}, \mathbf{z}_i)))) \tag{4.4}$$

We use Clipped Double Q-learning as proposed in (**?** ]) to further reduce overestimation bias during training. We also use Prioritized Experience Replay (**?** ]) to accelerate the training in sparse-reward tasks like AntMaze and FrankaKitchen. We summarize our proposed LDCQ method in Algorithm 3.

**Policy Execution**

The final policy for LDCQ comprises generating candidate latents $\mathbf{z}$ for a particular state $\mathbf{s}$ using the latent diffusion prior $\mathbf{z} \sim p_\psi(\mathbf{z} \mid \mathbf{s})$. These latents are then scored using the learnt Q-function and the best latent $\mathbf{z}_{max}$ is decoded using the VAE autoregressive decoder $\mathbf{a} \sim \pi_\theta(\mathbf{a} \mid \mathbf{s}, \mathbf{z}_{max})$ to obtain H-length action sequences which are executed sequentially. Note that the latent diffusion model is used both during training the Q-function and during the final evaluation phase, ensuring that the sampled latents do not go out-of-support.

### 4.3.3 Latent Diffusion Goal Conditioning (LDGC)

Diffuser (Janner et al. [29]) proposed framing certain navigation problems as a sequence inpainting task, where the last state of the diffused trajectory is set to be the goal during sampling. We can similarly condition our diffusion prior on the goal to sample from feasible latents that lead to the goal. This prior is of the form $p_\psi(\mathbf{z} \,|\, \mathbf{s}_0, \mathbf{s}_g)$, where $\mathbf{s}_g$ is the target goal state. Since with latent diffusion, the training of the low-level policy alongside the VAE is done separately from the diffusion prior training, we can reuse the same VAE posterior to train different diffusion models, such as this goal-conditioned variant. At test time, we perform classifer-free guidance to further push the sampling towards high-density goal-conditioned latents. For tasks which are suited to goal conditioning, this can be simpler to implement and achieves better performance than Q-learning. Also, unlike Diffuser, our method does not need to have the goal within the planning horizon of the trajectory. This allows our method to be used for arbitrarily long-horizon tasks.

### 4.3.4 Latent Diffusion-Constrained Planning (LDCP)

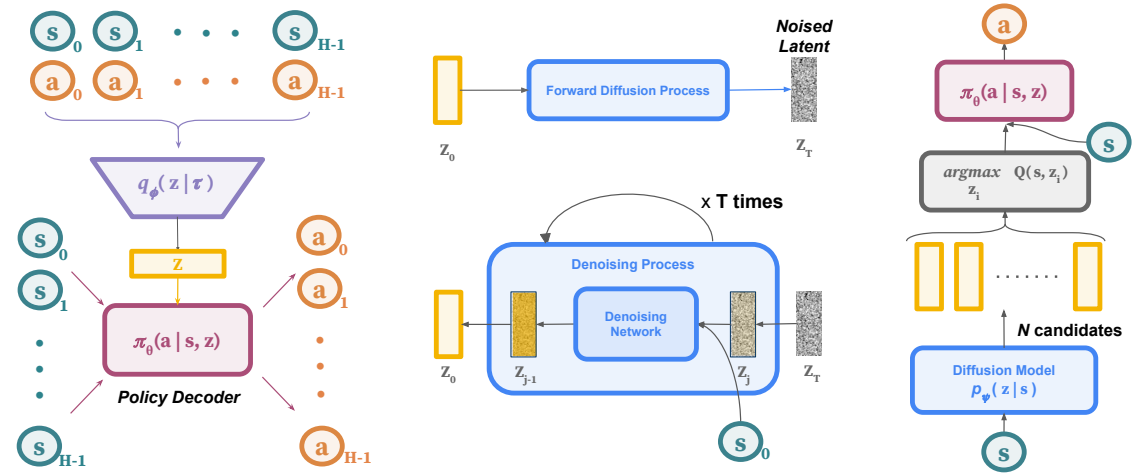
In this section, we explore another method to derive a policy for offline RL with latent diffusion. This is a model-based method which learns a temporally abstract world model of the environment with the offline data. Specifically, we learn a temporally abstract world model $p_\eta(\mathbf{s}_{t+H} \mid \mathbf{s}_t, \mathbf{z})$ that predicts the state outcome of executing a particular latent behavior after $H$ steps. That is, given the current state $\mathbf{s}_t$ and a latent behavior $\mathbf{z}$ the model predicts the distribution of the state $\mathbf{s}_H$. This is trained in a supervised manner by sampling transition tuples $(\mathbf{s}_t, \mathbf{z}, \mathbf{s}_{t+H})$ from $\boldsymbol{\tau}_H \sim \mathcal{D}$ and minimizing the objective:

$$\mathcal{L}(\eta) = \mathbb{E}_{\boldsymbol{\tau}_H \sim \mathcal{D}} \mid\mid p_\eta(\mathbf{s}_{t+H} \mid \mathbf{s}_t, \mathbf{z}) - \mathbf{s}_{t+H} \mid\mid^2 \tag{4.5}$$

where $p_\eta$ is the temporally abstract world model. In goal reaching environments, we leverage this model to do planning using the diffusion prior. We sample $n$ latents using the diffusion prior for a given state, $\mathbf{s}$ and use the learnt dynamics model to compute the final state $\mathbf{s}_H^i$ for each latent $\mathbf{z}^i$. These final states are then scored using a cost-function and the latent corresponding to the best final state is chosen for execution. We refer to this method as *Latent Diffusion-Constrained Planning*. The planning procedure is described in Algorithm 4.

**Algorithm 4** Latent Diffusion-Constrained Planning (LDCP)

**Input:** horizon $H$, number of latents to sample $n$, maximum iterations $M$, cost-function $\mathcal{J}$, policy decoder $\pi_\theta$, temporally abstract world model $p_\eta$, latent diffusion denoising function $\mu_\psi$, variance schedule $\alpha_1, \ldots, \alpha_T$, $\bar{\alpha}_1, \ldots, \bar{\alpha}_T$, $\beta_1, \ldots, \beta_T$.
*done* = *False*
**while** not *done* **do**
    Observe environment state $\mathbf{s}_0$
    Sample $n$ latents: $\mathbf{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    **for** $t = T$ to 1 **do** ▷ DDPM Sampling
        $\hat{\mathbf{z}} = \mu_\psi(\mathbf{z}_t, \varnothing, t) + w(\mu_\psi(\mathbf{z}_t, \mathbf{s}_0, t) - \mu_\psi(\mathbf{z}_t, \varnothing, t))$
        $\mathbf{z}_{t-1} \sim \mathcal{N}(\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{z}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\hat{\mathbf{z}}, \mathbb{I}(t > 1)\beta_t \mathbf{I})$
    **end for**
    Compute future states for each latent $\mathbf{z}_0^i$: $\mathbf{s}_H^i = p_\eta(\mathbf{s}_H^i \mid \mathbf{s}_0, \mathbf{z}_0^i)$
    Find best latent based on the cost-function: $i = \underset{i}{\text{argmin}}\ \mathcal{J}(\mathbf{s}_H^i)$
    Compute action-sequence using policy decoder $\pi_\theta(\mathbf{a} \mid \mathbf{s}_0, \mathbf{z}_0^i)$
    $h = 0$
    **while** $h < H$ **and** not *done* **do**
        Execute action $\mathbf{a}_h$
        Update *done*
        $h = h + 1$
    **end while**
**end while**




a) Latent Space Representation Training b) Latent Diffusion Model Training c) Policy Execution

Figure 4.1: **Latent Diffusion Reinforcement Learning Overview** a) We first learn the latent space and low-level policy decoder by training a $\beta$-VAE over $H$-length sequences from the demonstrator dataset. b) We train a latent diffusion prior conditioned on $\mathbf{s}_0$ to predict latents generated by the VAE encoder. c) After learning a Q function using LDCQ (Algorithm 3), we score latents sampled by the prior with this Q function and execute the low-level policy $\pi_\theta$ conditioned on the argmax latent.

## 4.4 Experimental Evaluation and Analysis

In our experiments, we focus on **1)** studying the effect of temporal abstraction on the latent space (section 4.4.1) **2)** understanding the need for diffusion to model the latent space (section 4.4.2 and 4.4.3) and **3)** evaluating the performance of our method in the D4RL offline RL benchmarks (section 4.4.4).

### 4.4.1 Temporal abstraction induces multi-modality in latent space

In this section, we study how the horizon length $H$ affects the latent space and provide empirical justification for learning long-horizon latent space representations. For our experiment, we consider the *kitchen-mixed-v0* task from the D4RL benchmark suite (Fu et al. [11]). The goal in this task is to control a 9-DoF robotic arm to manipulate multiple objects (microwave, kettle, burner and a switch) sequentially, in a single episode to reach a desired configuration, with only sparse 0-1 completion reward for every object that attains the target configuration. As Fu et al. [11] states, there is a high degree of multi-modality in this task arising from the demonstration trajectories because different trajectories in the dataset complete the tasks in a random order. Thus, before starting to solve any task, the policy implicitly needs to *choose* which task to solve and then generate the actions to solve the task. Given a state, the dataset can consist of multiple behavior modes, and averaging over these modes leads to suboptimal action sequences. Hence, being able to differentiate between these tasks is desirable.

We hypothesize that as we increase our sequence horizon $H$, we should see better separation between the modes. In Figure 4.2, we plot a 2D (PCA) projection of the VAE encoder latents of the starting state-action sequences in the kitchen-mixed dataset. With a lower horizon, these modes are difficult to isolate and the latents appear to be drawn from a Normal distribution (Figure 4.2). However, as we increase temporal abstraction from $H = 1$ to $H = 20$, we can see *three* distinct modes emerge, which when cross-referenced with the dataset correspond to the three common tasks executed from the starting state by the behavioral policy (microwave, kettle, and burner). These modes capture underlying variation in an action sequence, and having

picked one we can run our low-level policy to execute it. As demonstrated in our experiments, such temporal abstraction facilitates easier Q-stitching, with better asymptotic performance. However, in order to train these abstract Q functions, it becomes necessary to sample from the complex multi-modal distribution and the conventional VAE conditional Gaussian prior is no longer adequate for this purpose, as shown in section 4.4.2.

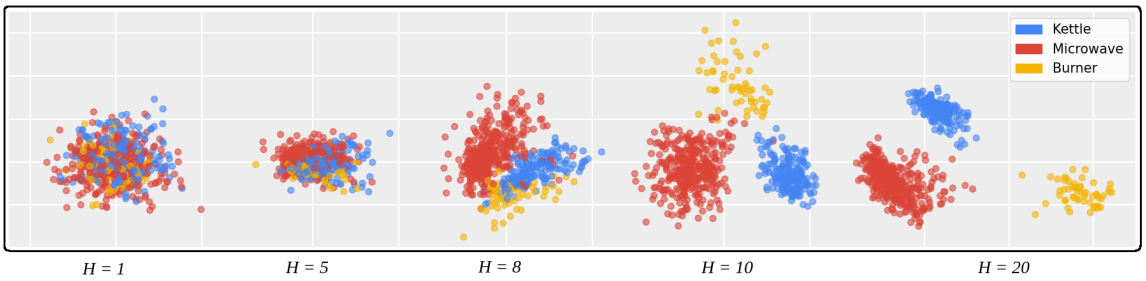
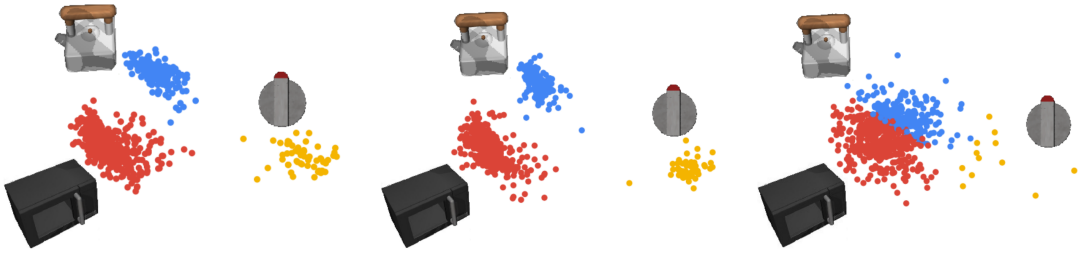



Figure 4.2: **Projection of latents across horizon**. Latent projections of trajectory snippets with different horizon lengths $H$. From the initial state there are 3 tasks (Kettle, Microwave, Burner) which are randomly selected at the start of each episode. These 3 primary modes emerge as we increase $H$, with the distribution turning multi-modal.

### 4.4.2 LDMs address multi-modality in latent space

In this section, we provide empirical evidence that latent diffusion models are superior in modelling multi-modal distributions as compared to VAEs. For our experiment, we again consider the *kitchen-mixed-v0* task. The goal of the generative model here is to learn the prior distribution $p(\mathbf{z} \mid \mathbf{s})$ and sample from it such that we can get candidate latents corresponding to state $\mathbf{s}$ belonging to the support of the dataset. However, as demonstrated earlier, the multi-modality in the latent spaces increases with the horizon. We visualize the latents from the initial states of all trajectories in the dataset in Figure 4.3 using PCA with $H = 20$. The three clusters in the figure correspond to the latents of three different tasks namely microwave, kettle and burner. Similarly, we also visualize the latents predicted by the diffusion model and the VAE conditional prior for the same initial states by projecting them onto the principal components of the ground truth latents. We can see that the diffusion prior is able to sample effectively all modes from the ground truth latent distribution,

while the VAE prior spreads its mass over the three modes, and thus samples out of distribution in between the three modes. Using latents sampled from the VAE prior to learning the Q-function can thus lead to sampling from out of the support, leading to extrapolation error.



(a) Ground truth (b) Diffusion prior (c) VAE prior

Figure 4.3: Visualization of latents projected using PCA for kitchen-mixed with $H = 20$. The diffusion prior models the ground truth much more accurately while the VAE prior generates out-of-distribution samples.

### 4.4.3 Performance improvement with temporal abstraction

We empirically demonstrate the importance of temporal abstraction and the performance improvement with diffusion on modelling temporally abstract latent spaces. We compare our method with a variant of BCQ which uses temporal abstraction ($H > 1$), which we refer to as BCQ-H. We use the same VAE architecture here as LDCQ, and fit the conditional Gaussian prior with a network having comparable parameters to our diffusion model. We find that generally, increasing the horizon $H$ results in better performance, both in BCQ-H and LDCQ, and both of them eventually saturate and degrade, possibly due to the limited decoder capacity. With $H = 1$, the latent distribution is roughly Normal as discussed earlier and our diffusion prior is essentially equivalent to the Gaussian prior in BCQ, so we see similar performance. As we increase $H$, however, the diffusion prior is able to efficiently sample from the more complex latent distribution that emerges, which allows the resulting policies to benefit from temporal abstraction. BCQ-H, while also seeing a performance boost with increased temporal abstraction, lags behind LDCQ. We plot D4RL score-vs-$H$ for BCQ-H and LDCQ evaluated on the *kitchen-mixed-v0* task in Figure 4.4.

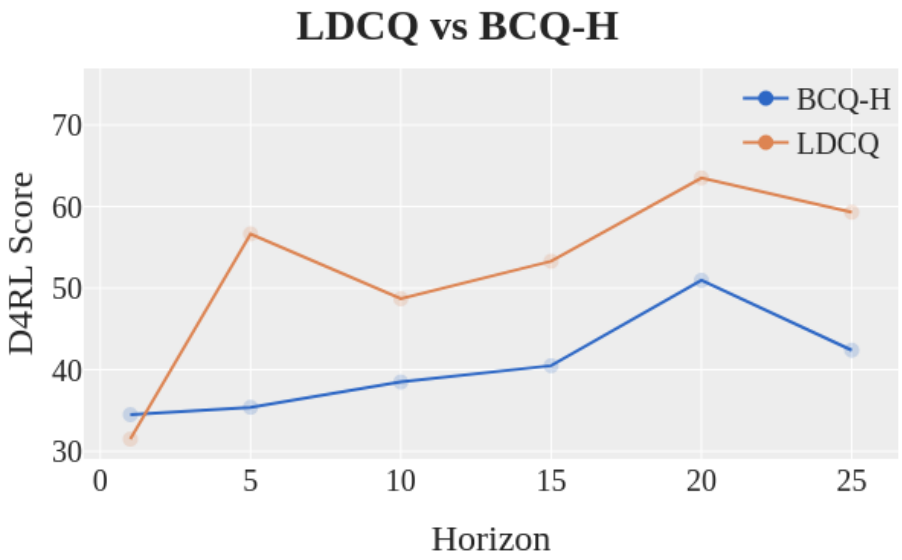



Figure 4.4: D4RL score of LDCQ and BCQ-H on kitchen-mixed-v0 with varying sequence horizon $H$

Table 4.1: Performance comparison on D4RL tasks which require long-horizon stitching with high multimodality. LDGC and LDCP variants are evaluated in the navigation environments.

| Dataset | BC | BCQ | CQL | IQL | DT | Diffuser | DD | LDCQ (Ours) | LDGC (Ours) | LDCP (Ours) |
|---|---|---|---|---|---|---|---|---|---|---|
| maze2d-large-v1 | 5.0 | 6.2 | 12.5 | 58.6 | 18.1 | 123.0 | - | **150.1** ± 2.9 | **206.8** ± 3.1 | **184.3** |
| antmaze-medium-diverse-v2 | 0.0 | 0.0 | 53.7 | **70.0** | 0.0 | 45.5 | 24.6 | **68.9** ± 0.7 | **75.6** ± 0.9 | **77.0** |
| antmaze-large-diverse-v2 | 0.0 | 2.2 | 14.9 | 47.5 | 0.0 | 22.0 | 7.5 | **57.7** ± 1.8 | **73.6** ± 1.3 | **59.7** |
| kitchen-partial-v0 | 38.0 | 31.7 | 50.1 | 46.3 | 42.0 | - | 57.0 | **67.8** ± 0.8 | - | - |
| kitchen-mixed-v0 | 51.5 | 34.5 | 52.4 | 51.0 | 50.7 | - | **65.0** | **62.3** ± 0.5 | - | - |

Table 4.2: Performance comparison on the D4RL locomotion tasks.

| Dataset | BC | BCQ | CQL | IQL | DT | Diffuser | DD | LDCQ (Ours) |
|---|---|---|---|---|---|---|---|---|
| halfcheetah-medium-expert-v2 | 55.2 | 64.7 | **91.6** | 86.7 | 86.8 | 88.9 | **90.6** | **90.2** ± 0.9 |
| walker2d-medium-expert-v2 | 107.5 | 57.5 | **108.8** | **109.6** | 108.1 | 106.9 | **108.8** | **109.3** ± 0.4 |
| hopper-medium-expert-v2 | 52.5 | **110.9** | 105.4 | 91.5 | 107.6 | 103.3 | **111.8** | **111.3** ± 0.2 |
| halfcheetah-medium-v2 | 42.6 | 40.7 | 44.0 | 47.4 | 42.6 | 42.8 | **49.1** | 42.8 ± 0.7 |
| walker2d-medium-v2 | 75.3 | 53.1 | 72.5 | 78.3 | 74.0 | 79.6 | **82.5** | 69.4 ± 3.5 |
| hopper-medium-v2 | 52.9 | 54.5 | 58.5 | 66.3 | 67.6 | 74.3 | **79.3** | 66.2 ± 1.7 |
| halfcheetah-medium-replay-v2 | 36.6 | 38.2 | **45.5** | **44.2** | 36.6 | 37.7 | 39.3 | 41.8 ± 0.4 |
| walker2d-medium-replay-v2 | 26.0 | 15.0 | **77.2** | 73.9 | 66.6 | 70.6 | 75.0 | 68.5 ± 4.3 |
| hopper-medium-replay-v2 | 18.1 | 33.1 | 95.0 | 94.7 | 82.7 | 93.6 | **100.0** | 86.2 ± 2.5 |

### 4.4.4 Offline RL benchmarks

In this section, we investigate the effectiveness of our Latent Diffusion Reinforcement Learning methods on the D4RL offline RL benchmark suite (Fu et al. [11]). We compare with Behavior Cloning and several *state-of-the-art* offline RL methods: Batch Constrained Q-Learing (BCQ) (Fujimoto et al. [13]), Conservative Q-Learing (CQL) (Kumar et al. [36]), Implicit Q-Learning (IQL) (Kostrikov et al. [34]), Decision Transformer (DT) (Chen et al. [5]), Diffuser (Janner et al. [29]) and Decision Diffuser (Ajay et al. [3]). The last two algorithms are previous trajectory diffusion methods. We found that our method does not require much hyperparameter tuning and only had to vary the sequence horizon $H$ across tasks. In maze2d and AntMaze tasks we use $H = 30$, in kitchen tasks we use $H = 20$ and in locomotion tasks we use $H = 10$. We train our diffusion prior with $T = 200$ diffusion steps. The other hyperparameters which are constant across tasks are provided in the Appendix.

In Table 4.1, we show results on the sparse-reward tasks in D4RL which require long horizon trajectory stitching. In particular, we look at tasks in Maze2d, AntMaze and FrankaKitchen environments which are known to be the most challenging in D4RL, with most algorithms performing poorly. Maze2d and AntMaze consist of undirected demonstrations controlling the agent to navigate to random locations in a maze. AntMaze is quite difficult because the agent must learn the high-level trajectory stitching task alongside low-level control of the ant robot with 8-DoF. In the maze navigation tasks, we also evaluate the performance of our goal-conditioned (LDGC) and planning (LDCP) variants. For Diffuser runs we use the goal-conditioned inpainting version proposed by the authors since the classifier-guided version yielded poor results. We found our implementation of BCQ improved over previous reported scores in kitchen tasks. Both our methods (LDCQ and LDGC) achieve state-of-the-art results in all sparse reward D4RL tasks. The goal-conditioned variant outperforms all others in maze2d and AntMaze. This variant is extremely simple to implement through supervised learning of the diffusion prior with no Q-learning or online planning and is ideal for goal-reaching tasks.

We also provide an evaluation of our method on the D4RL locomotion suite (Table 4.2). While these tasks are not specifically focused on trajectory-stitching, our method is competitive with other offline RL methods. We only run the LDCQ

variant here since they are not goal-reaching tasks.

## 4.5 Limitations

We list some limitations of our approach:

1. Our method, like other diffusion-based RL algorithms is slow at inference time due to the iterative sampling process, especially since we use a simple implementation of DDPM. This could be mitigated with methods that can perform faster sampling (Song et al. [61], Lu et al. [38], Dockhorn et al. [8], Xiao et al. [72]), or by distilling these diffusion models into others methods which need fewer sampling steps (Song et al. [63], Salimans and Ho [53]).
2. Our method has average performance on the locomotion task suite while having significant gains in the sparse reward tasks. We suspect the high periodicity of the walking gaits in the locomotion suite does not benefit much from reasoning with temporal abstraction. We also do not use a perturbation function during Q-learning like Fujimoto et al. [13], which makes it difficult for us to improve over the poor controllers in medium and medium-replay locomotion datasets. Introducing a perturbation function requires careful tuning to avoid extrapolation error, and the converged Q-learning wouldn't necessarily correspond to a high value policy, which is why other offline RL methods, which try to balance this tradeoff, evaluate online during training and consider the best scores. We however only evaluate a policy once after training is fully complete.
3. Another shortcoming of our work is that the sequence horizon $H$ for temporal abstraction has to be fixed for the entire experiment. We expect that being able to vary this adaptively could improve performance.

# Chapter 5

# Conclusions

We began by proposing an approach for simultaneously learning a set of skills, and a temporally abstract world model capable of predicting the long-term state transitions caused by those skills, purely from offline data. We have demonstrated that this world model enables sequences of skills to be planned rapidly online on an array of challenging, long time-horizon tasks. We then proposed using vector quantization to improve the quality of learnt latent variables for skill learning. These discrete skills learnt using VQ-VAEs are better suited for online optimization than continuous skills modeled with a standar VAE. Experiments in D4RL and Carla show that overoptimization causes problems with planning when using continuous latents. We hope future work finds engineering tricks to make training VQ-VAEs for skill learning simpler to implement and work consistently well across different environments.

Next, we moved to latent diffusion models which are significantly more expressive than standard VAEs, and have a much more stable optimization process than VQ-VAEs. We showed that offline RL datasets comprised of suboptimal demonstrations have expressive multi-modal latent spaces which can be captured with temporal abstraction and is well suited for learning high-reward policies. With a powerful conditional generative model to capture the richness of this latent space, we demonstrated that the simple batch-constrained Q-learning framework can be directly used to obtain strong performance. Our biggest improvements come from long-horizon sparse reward tasks, which most prior offline RL methods struggled with, even previous raw trajectory diffusion methods. Our approach also required

no task-specific tuning, except for the sequence horizon $H$. We believe that latent diffusion has enormous potential in offline RL and our work has barely scratched the surface of possibilities.

# Appendix A

# Model Architectures and Hyperparameters

We list the model architectures used for OPOSM, VQSkills and LDCQ here.

## A.1 OPOSM

### A.1.1 Model

For the VAE encoder, we use 2 stacked bidirectional GRU with 256 hidden units. It then has 2 heads which output the mean and standard deviation of the posterior. The conditional prior is a 2 layer MLP with 256 hidden units, and 2 output heads which each have 2 layers that produce the mean and standard deviation of the prior distribution. The standard deviation heads all have a SoftPlus activation to ensure they are positive. THe low level policy network and the TAWM are both MLPs that take as input the skill and current state and output the corresponding action and future state respectively. All networks use ReLU activations between hidden layers.

### A.1.2 Planning

We use CEM (Cross-Entropy Method) for planning. We use a skill planning depth of 10 skills. We do only 10 iterations of CEM so that we don't over-optimize, and find

that this is enough.

## A.2 VQSkills

### A.2.1 Model

The VAE encoder and decoder is identical to OPOSM described above. The prior is however instead a categorical distribution over the discrete latents in the codebook. For our best experiments in FrankaKitchen and AntMaze, The size of the codebook $k = 8$, and the number of discrete latents per skill $n = 4$.

### A.2.2 Planning

We use the same planning parameters as OPOSM, however we use the modified CEM planner as described in 3.4.2.

## A.3 LDCQ

### A.3.1 Model

The VAE encoder and decoder is identical to OPOSM described above, however while training we now use $\beta = 1e - 3$ instead of $\beta = 1$.

The diffusion model is a resnet architecture with 8 layers. It takes as input the current timestep noisy latent $z_t$, the state $s$ and diffusion timestep $t$. We use a linear variance schedule during diffusion model training. During classifer-free guidance while sampling, we set the weight $w = 0.5$. We use $T = 100$ diffusion steps for all experiments.

The Q networks for LDCQ are simple 4 layer MLPs with 4 layers. There is a LayerNorm applied before each activation as this has been found to help train value functions. There are 128 units in the hidden layers. We use prioritized experience replay buffer during Q-learning. We set the inverse temperature $\alpha = 0.7$ and importance weight parameter $\beta$ is linearly from 0.3 to 1.0 throughout training.

# Bibliography


[1] Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms, 2018.

[2] Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. Opal: Offline primitive discovery for accelerating offline reinforcement learning. In *International Conference on Learning Representations*.

[3] 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 *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=sP1fo2K9DFG.

[4] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019. URL http://arxiv.org/abs/1912.06680.

[5] 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. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[6] Felipe Codevilla, Eder Santana, Antonio Lopez, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9328–9337, 2019. doi: 10.1109/ICCV.2019.00942.

[7] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, and Martin Riedmiller. Magnetic control of tokamak plasmas through deep reinforce-

ment learning. *Nature*, 602:414–419, 02 2022. doi: 10.1038/s41586-021-04301-9.

[8] Tim Dockhorn, Arash Vahdat, and Karsten Kreis. Score-based generative modeling with critically-damped langevin diffusion. In *International Conference on Learning Representations*.

[9] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning diverse skills without a reward function. 2018.

[10] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022. doi: 10.1038/s41586-022-05172-4.

[11] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

[12] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/fujimoto18a.html.

[13] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062, 2019.

[14] Seyed Kamyar Seyed Ghasemipour, Dale Schuurmans, and Shixiang Shane Gu. Emaq: Expected-max q-learning operator for simple yet effective offline and online rl. In *International Conference on Machine Learning*, pages 3682–3691. PMLR, 2021.

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[16] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/haarnoja18b.html.

[17] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*.

[18] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

[19] Tiankai Hang, Shuyang Gu, Chen Li, Jianmin Bao, Dong Chen, Han Hu, Xin Geng, and Baining Guo. Efficient diffusion training via min-snr weighting strategy, 2023.

[20] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International conference on learning representations*.

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

[22] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf.

[23] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pages 6840–6851, 2020.

[24] Zhe Huang. Distributed reinforcement learning for autonomous driving. Master's thesis, Carnegie Mellon University, Pittsburgh, PA, May 2022.

[25] Minyoung Huh, Brian Cheung, Pulkit Agrawal, and Phillip Isola. Straightening out the straight-through estimator: Overcoming optimization challenges in vector quantized networks, 2023.

[26] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*.

[27] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, 2019.

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

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

[30] John M. Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A A Kohl, Andy Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David A. Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583 – 589, 2021.

[31] Heewoo Jun and Alex Nichol. Shap-e: Generating conditional 3d implicit functions, 2023.

[32] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[33] Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. Diffwave: A versatile diffusion model for audio synthesis. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=a-xFK8Ymz5J.

[34] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*.

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

[36] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1179–1191. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/0d2b2061826a5df3221116a5085a6052-Paper.pdf.

[37] Sergey Levine, Aviral Kumar, G. Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *ArXiv*, abs/2005.01643, 2020.

[38] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps, 2022.

[39] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11461–11471, June 2022.

[40] Marlos C Machado, André Barreto, Doina Precup, and Michael Bowling. Tem-

poral abstraction in reinforcement learning with the successor representation. *Journal of Machine Learning Research*, 24(80):1–69, 2023.

[41] Timothy Mann and Shie Mannor. Scaling up approximate value iteration with options: Better policies with fewer iterations. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 127–135, Bejing, China, 22–24 Jun 2014. PMLR. URL https://proceedings.mlr.press/v32/mann14.html.

[42] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[43] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[44] Judea Pearl. Causal inference in statistics: An overview. 2009.

[45] Karl Pertsch, Youngwoon Lee, and Joseph J. Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on Robot Learning (CoRL)*, 2020.

[46] Karl Pertsch, Youngwoon Lee, Yue Wu, and Joseph J Lim. Demonstration-guided reinforcement learning with learned skills. In *5th Annual Conference on Robot Learning*, 2021. URL https://openreview.net/forum?id=JSC4KMlENqF.

[47] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022.

[48] Ali Razavi, Aäron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2.

[49] 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 (CVPR)*, pages 10684–10695, June 2022.

[50] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[51] Chitwan Saharia, Jonathan Ho, William Chan, Tim Salimans, David J Fleet, and Mohammad Norouzi. Image super-resolution via iterative refinement.

*arXiv:2104.07636*, 2021.

[52] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Raphael Gontijo-Lopes, Burcu Karagol Ayan, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=08Yk-n5l2Al.

[53] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=TIdIXIpzhoI.

[54] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model, 2019. URL http://arxiv.org/abs/1911.08265. cite arxiv:1911.08265.

[55] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[56] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657*, 2019.

[57] Archit Sharma, Michael Ahn, Sergey Levine, Vikash Kumar, Karol Hausman, and Shixiang Gu. Emergent real-world robotic skills via unsupervised off-policy reinforcement learning, 2020.

[58] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016. URL http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

[59] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676): 354–359, 2017.

[60] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on*

*Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2256–2265, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/sohl-dickstein15.html.

[61] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=St1giarCHLP.

[62] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems*, pages 11895–11907, 2019.

[63] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models, 2023.

[64] 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.

[65] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

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

[67] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, page 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

[68] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6309–6318, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[69] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[70] Adam R Villaflor, Zhe Huang, Swapnil Pande, John M Dolan, and Jeff Schneider. Addressing optimism bias in sequence modeling for reinforcement learning. In *International Conference on Machine Learning*, pages 22270–22283. PMLR, 2022.

[71] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, L. Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Caglar Gulcehre, Ziyun Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, pages 1–5, 2019.

[72] Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. Tackling the generative learning trilemma with denoising diffusion GANs. In *International Conference on Learning Representations (ICLR)*, 2022.

[73] Wenxuan Zhou, Sujay Bajracharya, and David Held. Latent action space for offline reinforcement learning. In *Conference on Robot Learning*, 2020.