Exploring Reinforcement Learning approaches for Safety Critical Environments

Shivesh Khaitan

CMU-RI-TR-23-28

June 22, 2023

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

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

Abstract

Reinforcement Learning (RL) has emerged as a powerful paradigm for addressing challenging decision-making and control tasks. By leveraging the principles of trial-and-error learning, RL algorithms enable agents to learn optimal strategies through interactions with an environment. Over the years, RL has achieved remarkable successes in various domains, ranging from game playing to robotics and beyond. However, despite the success of these RL algorithms, their practical application in the real world still faces several challenges such as sample inefficiency and the lack of interpretability arising from the reactive nature of RL policies. RL algorithms require a substantial number of interactions with the environment to learn effective policies. This limitation hinders the applicability of RL in scenarios where data collection is expensive or time-consuming. In some environments, this data collection can also be potentially unsafe. Interpretability is crucial for understanding and trusting the decisions made by RL agents. RL algorithms are regarded as black box reactive policies, making it challenging to interpret the decision-making process, understand the factors influencing agent behavior, or even carry out safety checks before executing any commands. The impact of these challenges are aggravated in safety-critical environments, which includes many application areas within robotics. This work tries to address these challenges in the practical application of RL, particularly in safety-critical environments involving robotics applications. Overcoming these challenges will facilitate the adoption of RL in real-world settings, enabling intelligent decision-making and control in safety-critical domains.

Acknowledgments

I would like to express my sincere gratitude to Professor John M Dolan, my advisor for his consistent support and invaluable guidance throughout. His constant feedback and encouragement have been instrumental in shaping my research. His mentorship has had a profound impact on my academic and personal growth, and I am truly honored to have had the opportunity to work under his supervision.

I am immensely grateful to Professor Qin Lin as well who played a pivotal role in guiding me during my time at CMU as a RISS student. Under his mentorship, I was able to establish a strong foundation for my research journey.

Additionally, I would like to extend sincere appreciation to the members of my thesis committee Professor Jeff Schneider and Simin Liu, whose valuable feedback and insightful perspectives have greatly contributed to the refinement of my work. I am grateful for their dedicated time and expertise in reviewing my research and providing constructive suggestions.

I would like to thank my research colleagues, Siddarth Venkatraman and Ravi Tej Akella with whom I collaborated on my final project. I would also like to extend my gratitude to the entire research group under Professor Dolan for their diverse perspectives and collective expertise which have enriched my research experience and contributed to the overall success of this work.

I attribute my achievements to the unwavering support of my family, whose presence and encouragement have been crucial in my journey thus far. Without their steadfast support, I would not have been able to reach this significant milestone.

Funding

This work was supported by the CMU Argo AI Center for Autonomous Vehicle Research.

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

1.1 Overview

Reinforcement Learning (RL) has emerged as a highly influential and effective paradigm for tackling complex decision-making and control tasks. With their ability to harness the principles of trial-and-error learning, RL algorithms empower agents to acquire optimal strategies by engaging in iterative interactions with their environments. RL has been applied to a wide range of domains, including robotics, game playing, finance, and healthcare. In robotics, RL has been used to train robots for tasks such as object grasping and navigating challenging environments [4, 40]. In game playing, RL algorithms have achieved remarkable successes, surpassing human-level performance in games like chess and Go [45, 58, 61].

RL is particularly well suited for scenarios where explicit instruction or labeled data is either scarce or non-existent. Instead of relying on pre-defined rules or expert guidance, RL agents learn from experience, continuously refining their behavior to maximize rewards or minimize costs. This process closely mimics how humans and other intelligent beings learn through trial and error.

RL shows huge potential for complex robotic decision making tasks. These tasks are often safety-critical, which can have severe consequences of incorrect actions while training or testing RL agents in the real world. Recent research has focused on developing RL methods that prioritize safety and avoid hazardous actions. For example, the Safe Policy Improvement (SPI) algorithm incorporates safety constraints during policy optimization to ensure that the learned policies adhere to safety requirements [38]. Another approach, Constrained Policy Optimization (CPO), uses trust region optimization to enforce safety constraints and prevent unsafe actions during RL training [1]. These safety-aware RL techniques provide a valuable foundation for developing intelligent systems that can make reliable and safe decisions in critical domains.

Application of RL in safety-critical environments faces significant challenges. Training agents using RL requires a lot of training data in the form of environment interactions. However, data collection for robotics tasks is very expensive and can also be unsafe, especially when the tasks involve other dynamic agents in the environment, for example in autonomous driving tasks. Addressing sample inefficiency calls for the development of novel algorithms and techniques that can leverage data more efficiently, allowing agents to learn better and optimal policies with fewer samples. Previous works have explored techniques for improving sample efficiency in RL agents. Rainbow [26], a combination of improvements in deep RL, incorporates various techniques such as prioritized experience replay and distributional reinforcement learning to enhance sample efficiency. Additionally, Hindsight Experience Replay (HER) [6] enables RL agents to learn from failures by replaying experiences with different goals, allowing them to learn more efficiently from suboptimal outcomes. Curriculum Learning [9] is an approach that aims to improve sample efficiency by guiding the learning process through a curriculum of increasingly challenging tasks or environments. By strategically sequencing the learning process, curriculum RL offers a promising avenue to enhance sample efficiency and learn more effective policies while avoiding local optima in training. These advancements and techniques contribute to mitigating the sample efficiency problem in RL, enabling agents to learn effective policies with reduced data requirements. In chapter 2, we explore a novel curriculum learning technique for reinforcement learning in autonomous robotics tasks involving dynamic agents.

Interpretability is an important aspect of reinforcement learning (RL), as it enables us to understand and trust the decisions made by RL agents. Advancements in interpretable RL algorithms and techniques are necessary to provide transparent and interpretable decision-making in safety-critical applications. Reinforcement learning (RL) with model-based approaches [46] has gained attention as a potential solution for improving interpretability in RL. By incorporating an explicit model of the environment, model-based RL enables a structured representation of the underlying dynamics and transitions between states. This explicit model can provide insights and interpretability into the agent's decision-making process and avoid the problems of a completely reactive policy. Certain model-based methods are also more sample-efficient, as the learnt model can be used for improving policies without new environment interactions. Hierarchical learning [8] also tries to enable more structure to the solution and allows for an interpretable decision-making process. In chapter 3, we explore a method to combine a conventional motion primitives-based method for planning with reinforcement learning which uses a model for generating primitives and enables a high-level trajectory planning framework for the autonomous driving task.

Apart from being expensive and time-consuming, sometimes real-world data collection for RL agents can be entirely impossible. In such cases, offline reinforcement learning offers a promising approach to learning policies from static datasets. These offline reinforcement learning approaches [19, 35, 37] can learn effective policies from data collected by other agents, which can be human experts or any suboptimal controller. Developing safe and efficient offline RL methods that can leverage existing data without relying heavily on real-world interactions is essential for practical deployment in safety-critical environments. Previous methods have been unsuccessful in long-horizon credit assignment with offline datasets and encounter the problem of going out of support from the dataset, leading to poor agents. In chapter 4, we explore a novel method for offline reinforcement learning by leveraging powerful generative models. This method, while being able to learn from offline demonstrations, also allows a model-based approach, which is suitable for safety-critical systems.

To summarize, this research investigates innovative approaches to tackle important challenges in reinforcement learning, with a specific emphasis on safety-critical robotic tasks. We first propose the novel curriculum for improving sample efficiency which performs well, but is still a reactive planner. Next we investigate a RL based method to generate trajectory sequences using motion primitives. Finally, we move to a novel offline RL method which does not require any real-world interactions and is able to generate trajectory sequences which can be used for safety checks with the learnt model. This is highly suitable for safety critical systems. While aiming to improve sample efficiency and get interpretable trajectories, we also look into the real-time performance and optimality of these methods as these are crucial for final deployment of the agents.

1.2 Preliminaries

The reinforcement learning (RL) problem can be formulated as a Markov decision process (MDP). This MDP is a tuple $\langle \rho_0, \mathcal{S}, \mathcal{A}, r, P, \gamma \rangle$, where ρ_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, i.e., a mapping from states to actions, that maximizes the expected cumulative discounted reward.

1.3 Contributions

In this work, we explore three novel methods for reinforcement learning in safety-critical robotic tasks. As shown in fig. 1.1, the different methods have different desirable properties which usually requires trade-offs.

1. **State Dropout-based Curriculum Reinforcement Learning:** In chapter 2, we present a novel curriculum reinforcement learning method which improves an RL agent's sample efficiency and learns better policies as compared to a nominal agent trained without our proposed curriculum.
2. **Motion Primitives-based Reinforcement Learning:** In chapter 3, we present a hybrid method combining motion primitives with reinforcement learning for decision making which can be used for generating longer trajectory sequences (as opposed to reactive policies) which are desired for safety checks.
3. **Offline Reinforcement Learning with Latent Diffusion:** In chapter 4, we present a novel offline reinforcement learning method for robotic control tasks which learns policies for agents without any real-world interactions. This

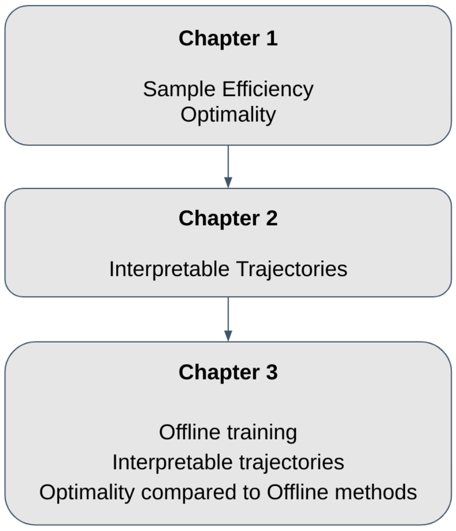
Chapter 1
Sample Efficiency
Optimality
Chapter 2
Interpretable Trajectories
Chapter 3
Offline training
Interpretable trajectories
Optimality compared to Offline methods

Figure 1.1: Desirable properties of the proposed methods.

method also allows for learning temporally extended dynamic models which can be used for obtaining interpretable solutions. This is crucial for enabling certain intelligent agents for safety critical systems.

In essence, this work explores improvements in reinforcement learning for safety-critical robotic tasks which offer a powerful framework for addressing decision-making and control problems contributing to the development of more intelligent and adaptive reinforcement learning agents.

Chapter 2

State Dropout-based Curriculum Reinforcement Learning

2.1 Introduction

A major drawback of RL methods is long training periods requiring a lot of environment interactions and convergence to sub-optimal policies. This prohibits the usage of RL in safety-critical systems, as the systems might be required to enter unsafe states while the agent trains. To accelerate the training process, curriculum learning [9] has been proposed. The basic idea in curriculum learning is to design the training process as a set of graduated steps with increasing complexity of tasks. It draws inspiration from the way humans learn by starting with easier tasks and gradually increasing the complexity to become an expert. The idea of curriculum learning has also been applied to reinforcement learning. For a detailed review on the applications of curriculum for reinforcement learning, see [47]. Apart from accelerating the training process, it has also been proposed that using a curriculum can help find better local minima as compared to training without curriculum [9].

Previously, most works have used hand-designed task-based curricula for training, which rely on segregating the tasks at hand based on their difficulty and training sequentially with increasing levels of difficulty. Such manual segregation of tasks can be laborious and time-consuming. In this work, we propose a unique automated curriculum for training, that is easy to design and is also more effective. Our curriculum is specifically focused on environments with other dynamic actors involved. In these environments, learning the dynamics of other agents possess a significant challenge and it takes a sufficient number of interactions with the environment before the agent can understand the dynamics to improve its policy. Our proposed curriculum makes it easier for the RL agent to learn the environment dynamics. For our experiments, we consider an unsignalized intersection traversal task in autonomous driving on the CommonRoad-RL benchmark [71].

The rest of this chapter is organized as follows. Section 2.2 provides a review of some important related work. Section 2.3 gives an introduction to the intersection problem addressed in this work. Section 2.4 presents the proposed curriculum. Section 2.5 presents the experimental results. The conclusion is in Section 2.6.

2.2 Related Work

This section summarizes relevant previous work which proposes techniques for accelerating training for learning agents.

2.2.1 Transfer Learning

Transfer learning (TL) is a technique which pretrains a network on a different task before training on the target task. The network benefits by using the skills acquired during pretraining. For a comprehensive literature review on transfer learning for RL, see [72]. Apart from just having the advantage of faster training time, using transfer learning also leads to better generalization and better performance than an agent trained without using the pretrained network.

2.2.2 Curriculum Learning

Curriculum learning [9] draws inspiration from how humans learn to do complex tasks by “starting small” [16]. The main idea is that once the agent has learned to do a simple task well, it will take fewer iterations to adapt to a harder task having the simpler task as a subset because of the stability of the training process. Apart from just accelerating the training process, it has also been shown that certain tasks have been nearly impossible to train without following a curriculum [4]. [64] used curriculum reinforcement learning for overtaking in an autonomous racing scenario. However, hand-segregating tasks for designing a curriculum might be cumbersome, as deciding the difficulty level of a task might not be trivial. [50] proposes an automatic curriculum generator that tries to alleviate the problem.

In this work, we propose novel automated curricula for autonomous driving at unsignalized intersections. The curricula are easier to train and reduce the training time while converging to better policies.

2.3 Problem Definition

2.3.1 Problem Statement

The problem which we consider is generating control commands for an ego-vehicle to traverse unsignalized intersections and reach a predefined goal state. We consider two types of intersection scenarios, as shown in Fig. 2.1 and Fig. 2.2. The ego-vehicle is depicted in green, target vehicles are in blue and the goal region is in yellow. The dotted lines starting from target vehicles show their future trajectories, which do not change in response to the ego-vehicle behavior. This increases the difficulty level of the problem significantly.

T-intersection These scenarios are adopted from the CommonRoad benchmarks: `ZAM_Tjunction` (Fig. 2.1).

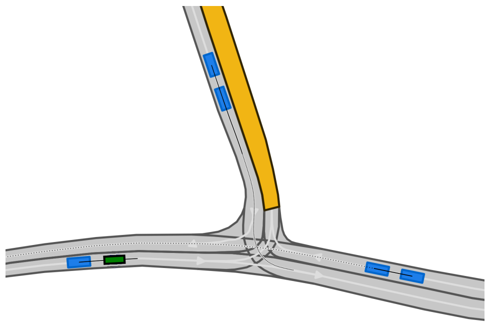

Figure 2.1: Visualization of T-intersection scenarios.

Four-way intersection These are four-way intersection scenarios, (Fig. 2.2) custom-generated using the CommonRoad scenario designer tool [60]. The goal state for these scenarios can be in the left, straight-ahead, or right lane. The ego-vehicle has to reach the goal region within $\pm 11.45°$ of the road lane orientation to successfully complete a scenario. The limits on orientation ensure that the ego-vehicle does not reach the goal with an arbitrary orientation, which can make it harder for the vehicle to correct the orientation when moving further in the goal lane. The scenario has to be completed before a time-out of 150 sec.

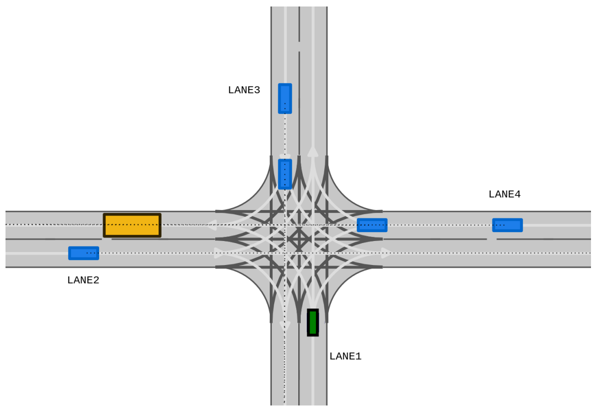

Figure 2.2: Visualization of Four-way intersection scenarios.

2.3.2 Observation and Action space

The observation space for the RL agent is as defined in Table 2.1. The observations can be divided into four categories: 1) ego-vehicle state; 2) goal and reference path-related observations; 3) surrounding vehicles-related observations; and 4) road network-related observations. The dynamics constraints are based on the kinematic bicycle model (`KS1`) as described in [5]. The reference path observation consists of waypoints from the path generated by A* search over the lanelet network in Commonroad. The distance advancements for longitudinal and lateral directions are calculated along this reference path. For the surrounding vehicles, we include the current and future states (for 5 time-steps with a time discretization of 0.1 sec) of each of the five vehicles in the ego-vehicle frame. The information in future states is as described in section 2.4.

The action space for the agent consists of acceleration and steering for the ego-vehicle $A = \{accel, steer\}$. The controls are continuous.

Table 2.1: Observation Space for the RL agent

Variable	**Description**	**Values**
v_t	Ego-vehicle absolute velocity	$\mathbb{R}$
a_{t-1}	Ego-vehicle previous acceleration	$\mathbb{R}$
δ_{t-1}	Ego-vehicle previous steering	$\mathbb{R}$
θ_t	Ego-vehicle orientation	$\mathbb{R}$
r	Reference path waypoints (x, y, θ) in ego-vehicle frame	$\mathbb{R}^{15}$
$d\theta_t$	Ego-vehicle orientation deviation from reference path	$\mathbb{R}$
d_{long}	Longitudinal distance advancement towards goal	$\mathbb{R}$
d_{lat}	Latitudinal distance advancement towards goal	$\mathbb{R}$
t_{out}	Time remaining before time-out	$\mathbb{R}$
c	Lane curvature	$\mathbb{R}$
l_o	Ego-vehicle offset from lane centerline	$\mathbb{R}$
r_l	Ego-vehicle distance from left road boundary	$\mathbb{R}$
r_r	Ego-vehicle distance from right road boundary	$\mathbb{R}$
l_l	Ego-vehicle distance from left lane boundary	$\mathbb{R}$
l_r	Ego-vehicle distance from right lane boundary	$\mathbb{R}$
o	Target vehicles' current and future states$^*(x, y, v, \theta)$	$\mathbb{R}^{100}$

$*$ The information in future states depends on the curriculum

2.3.3 Reward Structure

The components of the reward function are as follows:

- A positive reward for distance advancement along the reference path as described in section 2.3.2. The reward is calculated as $\rho_1 d_{long} + \rho_2 d_{lat}$ where ρ_1 and ρ_2 are positive constants.
- A constant negative reward σ_1 for collision with obstacles, going off-road, and violating dynamics constraints.
- A constant negative reward σ_2 for time-out.
- A constant positive reward ϕ for reaching the goal within the allowed orientation error.

An episode is terminated when any one of the following conditions is met: ego-vehicle reaches the goal within allowed orientation error, ego-vehicle goes off-road, ego-vehicle collides with another obstacle, applied controls violate dynamics constraints or time-out.

2.4 State Dropout Curriculum

Our curriculum is based on the idea that when the dynamics of the environment are known, it is easier for the RL agent to learn a policy. The RL agent can have access to this information from the observation space where the future trajectories of the dynamic actors in the environment can be incorporated as observations. However, in practice, getting this privileged information is not possible during testing. Nevertheless, it is possible to get this information during the training phase through simulations or artificially created real world testing environments. Our curriculum incorporates this future trajectory information about dynamic actors into the observation space and learns a policy to complete the task. Gradually, it learns to complete the task without having access to the privileged information by *forgetting* to use it through *State Dropout*. We propose two curricula to *forget* this privileged information while training such that by the end of the training phase, the agent can complete the task without any privileged information.

Algorithm 1 Step function for curriculum 1

```
1: procedure STEP(action, phase)
2:     Apply acceleration and steering from action
3:     obs = Observe()
4:     reward, done = Reward(obs)
5:     for $i \leftarrow (N - phase + 1)$ to $N$ do
6:         Drop $i^{th}$ future prediction from obs
7:     end for
8:     return obs, reward, done
9: end procedure
```

2.4.1 Curriculum 1

In this method we train the agent in $N + 1$ phases starting with phase 0. We begin training with the future state information for N time-steps. After phase 0, we drop the N^{th} future state information and continue training. In the subsequent phases we keep on dropping more future state information and in the final phase fine-tune the agent without any future state information. Algorithm 1 describes the `STEP` function for this method.

2.4.2 Curriculum 2

In this method, instead of dropping the future state information sequentially after each phase, we augment the action space of the agent with an additional action such that the augmented action space is $A = \{accel, steer, pred\}$. Here *pred* determines which future state information for the surrounding vehicles will be dropped from the observation in the next step. The reward structure as defined in section 2.3.3 is also augmented to add constant positive rewards ψ_i for choosing to drop the future states from the i^{th} to the N^{th} time-step. The decision is based on parameters κ_i. Here, $\forall i \in [2, N]$.

$$\psi_i < \psi_{i-1} \tag{2.1}$$

$$\kappa_i < \kappa_{i-1} \tag{2.2}$$

Algorithm 2 Step function for curriculum 2

```
 1: procedure STEP(action)
 2:     Apply acceleration and steering from action
 3:     obs = Observe()
 4:     reward, done = Reward(obs)
 5:     for $i \leftarrow 1$ to $N$ do
 6:         if $action.pred \geq \kappa_i$ then
 7:             Drop $i^{th}$ future prediction from obs
 8:             reward = reward + $\psi_i$
 9:         end if
10:     end for
11:     return obs, reward, done
12: end procedure
```

This incentivizes the agent to drop more future states. Algorithm 2 defines the modified `STEP` function for this method. Thus we let the network choose when to drop the future states. At the beginning of training, when the agent has not learned to predict the behavior of the vehicles, it chooses to use the future information. As the training progresses and the agent learns to control the ego-vehicle, ψ_i incentivizes the agent to learn to predict the driving intention as well in order to get higher rewards. An added advantage of this method is that the *pred* value can be used at test time to determine whether the agent understands the behavior of the other vehicles or not. This can further be used as a confidence score for the agent commands.

2.5 Experiments

The curriculum was tested on the CommonRoad simulator for unsignalized T-intersections and four-way intersections. We use a PPO policy [51] for the agents for our testing; however, the curriculum can be used with other reinforcement learning methods as well.

2.5.1 Experimental Setup

For the T-intersections, we have 544 `ZAM_Tjunction` scenarios from the CommonRoad benchmarks, which have scenarios with ego-vehicles and target-vehicles initialized at different positions with different velocities. 380 scenarios are used for training, and the rest are used for testing.

We generated 2000 four-way intersection scenarios. For each scenario, the ego-vehicle is spawned at a randomly sampled location in lane 1 (Fig. 2.2) with appropriate orientation. The ego-vehicle always starts at rest. The obstacles are spawned at randomly sampled locations in lanes 2, 3 and 4 with appropriate orientation and randomly sampled initial velocities $\in [5, 10]$ m/s. 1400 scenarios were used for training and the rest were used for testing.

2.5.2 Network Architecture

To represent the policies, we used fully-connected (FC) networks with 8 hidden layers of 64 units each and *tanh* nonlinearities. The hyperparameter values for training the PPO agent are listed in table 2.2. The empirically best-performing hyperparameters and network architecture were chosen for the experimental results.

2.5.3 Training Details

The training was done on a desktop with 3.5 GHz AMD Ryzen 9 5900hs CPU. We used a customized version of the CommonRoad-RL framework for training and testing.

Table 2.2: Hyperparameters for training the RL agent

Parameter	**Value**
Training Iterations (T-intersections)	600
Training Iterations (Four-way intersections)	650
Discount (γ)	0.99
GAE parameter (λ)	0.95
Clipping parameter	0.2
Learning-rate	0.0005
No. of environments	32
Time horizon	512
Batch size	16384*

* Batch size = Time Horizon $\times$ No. of environments

We trained three different agents:

1. A standard PPO agent without any curriculum. The target vehicles' future state information is dropped by default in the standard agent.
2. A PPO agent using curriculum 1 with $N = 4$. The agent is thus trained in 5 phases.
3. A PPO agent using curriculum 2 with $N = 4$.

2.5.4 Performance Evaluation

To understand the impact of the proposed curricula, we compare the training curves of the agents trained using the curricula with the training curve of the standard PPO agent. We also compare the success rate and mean reward to evaluate the performance of our method. The results presented are the best results obtained by repeating each experiment 3 times with a random seed for the network. The future states of the surrounding vehicles are not available during testing for all three methods.

T-intersection

The training curves for the three methods are shown in Fig. 2.3. The rewards are averaged across 10 updates. It can be seen that both the curriculum-based methods outperform the standard PPO baseline in terms of sample efficiency. While PPO agents with the curricula start converging around 6000000 steps, the baseline standard PPO converges around 8250000 steps.

Fig. 2.4 shows the success rate of the methods on 164 testing scenarios. Both the curricula outperform the standard PPO baseline with their best performance of 99.39% as compared to 92.68% success rate for the baseline.

Four-way intersection

The training curves for the three methods are shown in Fig. 2.5. The rewards are averaged across 10 updates. It can be seen that both the curriculum based methods outperform the standard PPO baseline in terms of sample efficiency. While PPO

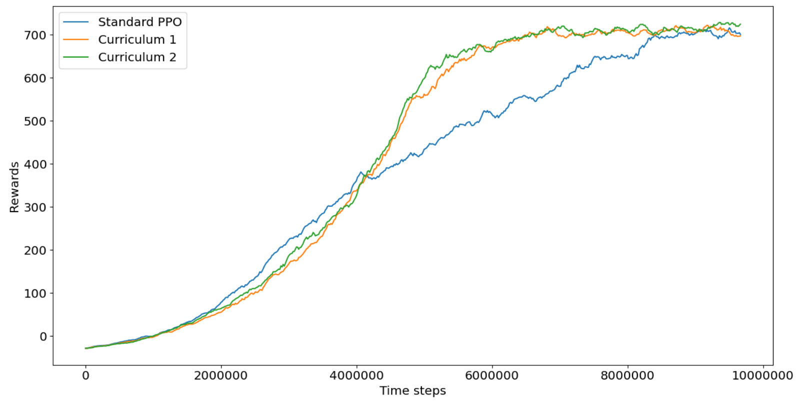

Figure 2.3: Training curves for T-intersection experiments

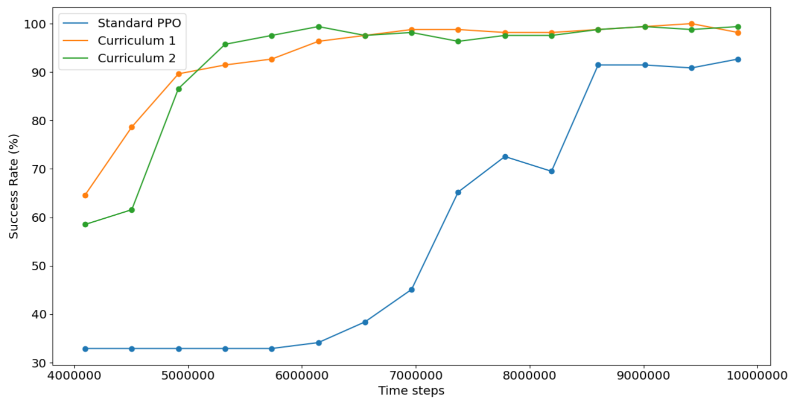

Figure 2.4: Success rates for T-intersection experiments

with curriculum 1 starts converging around 7250000 steps, the curriculum 2 approach converges around 8500000 steps. The baseline converges around 9250000 steps.

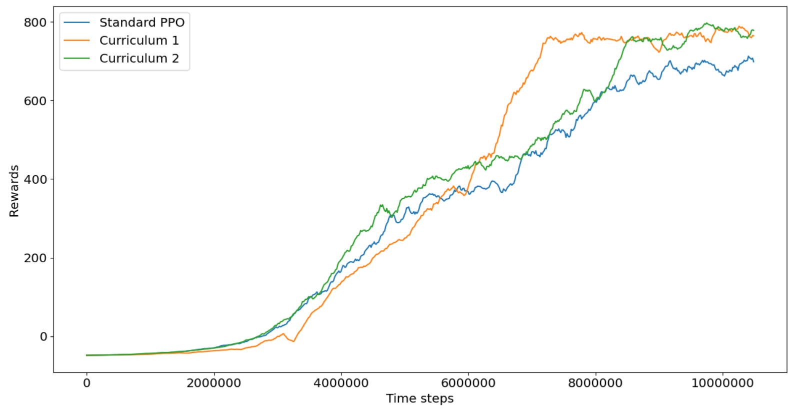

Figure 2.5: Training curves for four-way intersection experiments.

Fig. 2.6 shows the performance of the methods on the 600 testing scenarios. Curriculum 2 shows the best performance, with its best model having a success rate of 93.83%. Curriculum 1 achieves 92.66% and the baseline achieves 81.33%.

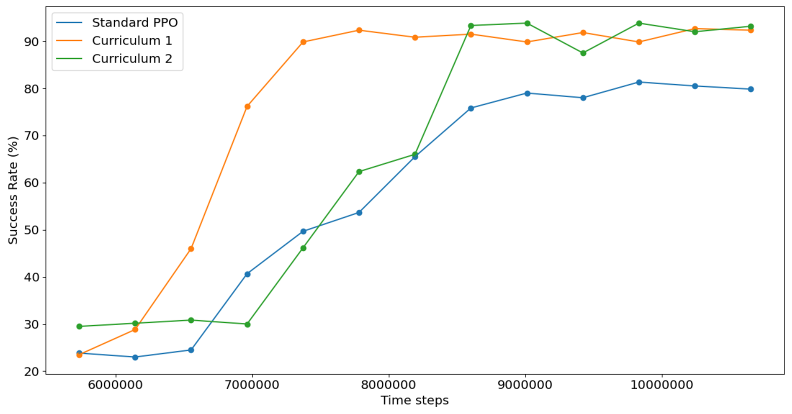

Figure 2.6: Success rates for four-way intersection experiments

We also compare the results with the rule-based TTC method [39] in Table 2.3.

2.5.5 Discussion

The training curves and success rates of the curriculum-based agents show that using the curriculum results in higher sample efficiency, with the agents converging much

Table 2.3: Comparison of success rates of methods with curriculum versus baselines

Method	**T-intersection (%)**	**Four-way intersection (%)**
TTC	90.85	84.33
Standard PPO	92.68	81.33
Curriculum 1	**99.39**	92.66
Curriculum 2	**99.39**	**93.83**

faster as compared to the baseline agent. Also, both the curricula have higher success rates than the baselines in both the scenarios. This shows that the agents converge to better optima as the curriculum makes the training stable. Though we used ground truth prediction for the target vehicles available in the simulation, for real-world training, predictions from the perception layer can be used. Further avenues of privileged information, e.g., bird's-eye view in the augmented observation space, can also be experimented with.

2.6 Conclusion

In this chapter, we explored a novel curriculum for training a deep reinforcement learning agent where future state predictions can be used to achieve faster training and avoid convergence to suboptimal policies. We test the performance of the curriculum on the unsignalized intersection traversal task for autonomous driving. The curriculum outperforms the standard baseline in sample efficiency and test time performance. Though in this work we show the application of the curriculum to the intersection traversal task only, it has a broader scope and can be generally applied to other safety-critical tasks as well.

Chapter 3

Motion Primitives-based Reinforcement Learning

3.1 Introduction

Reinforcement learning (RL) has been combined with conventional planning approaches to leverage the strengths of both paradigms. By integrating RL with classical planning techniques, the strengths of both approaches can be leveraged to achieve more effective and interpretable solutions. RL can handle complex and uncertain environments, while planning algorithms excel at reasoning and generating structured plans. The combination of RL and planning enables agents to learn from interactions with the environment while also using prior knowledge or domain-specific rules potentially reducing the amount of data or real-world interactions required for training. Monte Carlo Tree Search (MCTS) [10] has been used in RL agents for decision-making, allowing for some of the most advanced game-playing intelligent agents [61]. Hierarchical RL [8] involves decomposing complex tasks into subtasks or skills, enabling a more structured and interpretable representation of the agent's decision-making process. These hybrid approaches provide a synergistic combination of RL and conventional planning, leading to more interpretable and efficient solutions for decision-making tasks.

Such combination of reinforcement learning and planning approaches is highly beneficial for safety critical environments, where prior planning, hierarchical and model-based approaches can be used for increasing sample efficiency, incorporating safety, dynamic feasibility and improving overall policies.

Researchers in the past have experimented with several methods which generate high-level trajectories using learning-based approaches while the control commands are generated by a traditional control module. These trajectories can then be easily checked for collision before the execution of the control commands. Also these approaches allow for faster credit assignment, which is a major challenge in reinforcement learning [43, 44].

Motion Primitives are a popular choice for conventional planning in robotic tasks. This involves using offline pre-computed trajectories generated using an available model of the robot and sequencing optimal combinations of them online. The online sequencing often requires a graph-search in a spatio-temporal lattice constructed out of the offline primitives. The path selection is done using hand-engineered evaluation functions which can account for multiple objectives including safety. The complexity of the graph search limits the number of primitives that can be used, thereby limiting the performance of the planner. While several recent works have addressed the generation of optimal offline primitives, including data-driven methods for primitive generation, there has been limited work in improving the graph-search for constructing the trajectory online.

In this work, we propose a framework for generating high-level trajectories for robotic control using reinforcement learning. To the best of our knowledge, this is the first work which demonstrates how reinforcement learning can be used with large motion primitive libraries to generate interpretable trajectories. For our experiments, we consider an unsignalized intersection traversal task in autonomous driving on the CommonRoad-RL benchmark [71]. We demonstrate the effectiveness of our method by testing it using CommonRoad-RL [71] on unsignalized T-intersections and four-way intersections and compare it with the CommonRoad-Search planner, which uses conventional search for sequencing the optimal primitives and a conventional classification agent.

The rest of this chapter is organized as follows. Section 3.2 provides a review of some important related work. Section 3.3 gives an introduction to the problem formulation for motion primitive-based planning. Section 3.4 describes our primitives-based reinforcement learning method. Section 3.5 presents the experimental results. The conclusions are in Section 3.6.

3.2 Related Work

This section summarizes relevant previous work which combines reinforcement learning with conventional planning techniques.

3.2.1 Model-based Reinforcement Learning

Model-based reinforcement learning (RL) approaches have gained significant attention as a means to enhance sample efficiency and improve decision-making. By building an explicit model of the environment, these methods capture the dynamics and transitions between states, enabling agents to plan and simulate potential trajectories before taking actions. Different model-based approaches leverage the learnt model in different ways. For example, a class of methods use the model to artificially generate more training samples for the learning agent [22, 66], while [12] and [23] use the learned model during the actual execution phase, where the learned models are used to generating final control commands. These approaches ultimately enhance the performance of RL agents by increasing sample efficiency, or making the framework more interpretable.

3.2.2 Hierarchical Reinforcement Learning

By explicitly defining a hierarchy of goals and actions, interpretable policies can be constructed at different levels of abstraction [14]. These modular frameworks often use learning for generating high-level plans or trajectories to be tracked by low-level controllers [13, 30, 48]. This makes the system more transparent, as the high-level trajectory can be further optimized or checked for feasibility and collisions before execution. These trajectories are also more generalizable to different robotic systems and environments than an end-to-end controller. [69] proposed a classification agent which scores a set of primitives and chooses the best primitives using the output score. However, the size of the motion primitives library considered is very small, which is not scalable to highly dynamic robotic environments like autonomous driving.

In this work, we demonstrate how reinforcement learning can be used for generating such trajectories in a motion primitive-based planning framework consisting of a large motion primitives library. To the best of our knowledge, our work is the first to demonstrate how learning can be used for selecting the optimal primitive from among thousands of offline generated primitives. This has the advantage of being faster than traditional lattice search methods and still being interpretable and generalizable to changing environments without manual hand-tuning.

For our experiments, we use Proximal Policy Optimization (PPO) [59], which is a variant of the actor-critic family and uses an adaptive KL divergence penalty to control the change of policy at each iteration.

3.3 Problem Formulation

We consider the problem of goal reaching for a robotic system with dynamics defined as $\dot{x} = f(x, u)$ where $x \in \mathcal{X}$ and $u \in \mathcal{U}$. Here $\mathcal{X} \in \mathbb{R}^n$ represents the robot state space and $\mathcal{U} \in \mathbb{R}^m$ represents the control space. Along with the robot's state space, there exists uncontrolled environment E. A robot state and environment state combined is denoted as an observation $\mathcal{O}$. For a given initial condition the planner needs to generate a collision-free trajectory which drives the robot to the goal. We assume access to a library of motion primitives $\mathcal{L}$ constructed using the available dynamics model of the system. Every primitive $l \in \mathcal{L}$ is defined by a sequence of states $\{x_0, x_1, \ldots x_t\}$ such that the sequence is a dynamically feasible trajectory starting at x_0 and ending at x_t.

The planning problem which we consider is generating trajectories for an autonomous driving intersection traversal scenario. This was chosen because it is a highly dynamic scenario with narrow available drivable space and hence tests the limits of the primitives. However, this method can be generally applied to other robotic environments making use of motion primitives. The ego-vehicle has to traverse unsignalized intersections and reach a predefined goal state. We consider the intersection scenarios as described in section 2.3.1.

3.4 Primitives-based RL

We present a planning policy which maps the current state of the robot and environment (observation) to an optimal motion primitive $\mathcal{O} \rightarrow \mathcal{L}$. To deal with the large number of discrete actions, we use a modified version of the Wolpertinger architecture [15] as described in this section further. Fig. 3.1 shows an overview of our framework.

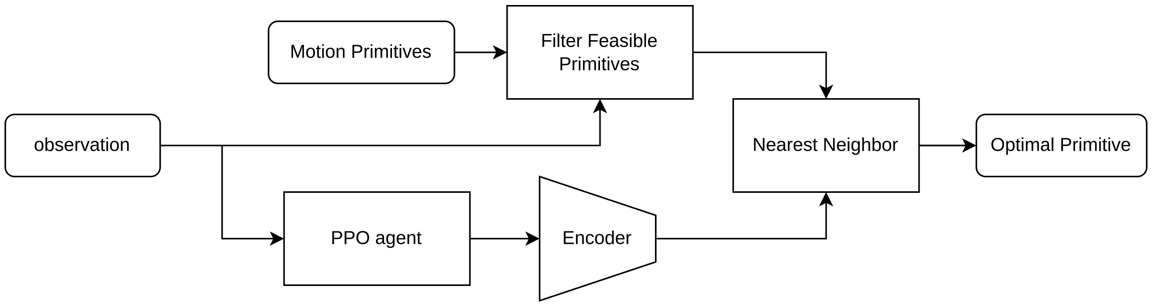

Figure 3.1: **Overview of our proposed framework.** The PPO agent maps the observation to an intermediate representation which is encoded by the encoder into a primitive embedding. The embedding's nearest neighbor from among the feasible primitives is selected as the optimal primitive for execution.

3.4.1 Motion Primitive Embeddings

We train an autoencoder to generate low-dimensional embeddings for the motion primitives. This gives us a mapping from $\mathcal{L} \rightarrow \mathcal{Z}$. Let $\mathcal{Z} \in \mathbb{R}^p$. Once trained, the encoder f_θ is used offline to generate embeddings for all the primitives in $\mathcal{L}$. The latent space representation for primitives allows the framework to deal with long-duration primitives.

3.4.2 Online Optimal Primitive Selection

The optimal primitive needs to be selected online during planning at each step. We learn a policy (π_ϕ) using PPO to select the primitive from the library of offline primitives. However, directly mapping the observation to a primitive with a discrete action space agent is difficult, as the primitive library is very large and only a small subset of primitives is feasible for a particular robot state. Thus, the PPO agent is trained to first map the observations ($\mathcal{O}$) to $\mathcal{Y}$ where $\mathcal{Y} \in \mathbb{R}^{nt}$. Thus

$$\pi_\phi : \mathcal{O} \to \mathcal{Y}$$

The agent's output is then encoded using the encoder from offline training. This gives us an optimal motion primitive embedding $\hat{z} \in \mathbb{R}^p$

$$f_\theta : \mathcal{Y} \to \mathbb{R}^p$$

This embedding is then compared with all the feasible primitive embeddings ($\mathcal{Z}'$) in $\mathcal{Z}$ and the nearest neighbor z of $\hat{z}$ is selected. The motion primitive corresponding to $\hat{z}$ is used for execution

$$\hat{l} = g(\arg\min_{z \in \mathcal{Z}'} \|z - \hat{z}\|_2)$$

where g maps the primitive embedding back to the corresponding primitive.

Comparing $\hat{z}$ with only the feasible motion primitive embeddings ensures that the selected motion primitive is always feasible for execution. We do not check for collision when filtering the primitives. Only dynamic feasibility from the current state is checked, as it is computationally inexpensive to compute. The feasibility of a primitive $l \in \mathcal{L}$ is decided by a weighted distance between the current state of the robot and the first state in the primitive. If the weighted distance is less than a threshold l_{thresh}, the trajectory is considered feasible. The parameter l_{thresh} can be decided based on the robustness of the controller. This filtering prunes the search space and allows us to use the nearest neighbor instead of computing k-nearest neighbor and training another Q-function as in the original Wolpertinger architecture.

Policy parameters ϕ is trained to maximize the rewards as described in section 3.4.4 using PPO. During training, the rewards are delayed until a primitive is executed. However, an episode can terminate while a primitive is getting executed if any of the termination criteria are satisfied. The encoder layers are frozen during the policy training. Algorithm 3 summarizes the step function for the planner, which receives an observation and selects and executes the primitive.

Algorithm 3 Step function for the agent

```
1: procedure STEP(obs : $\mathcal{O}$)
2:     Filter $\mathcal{Z}'$ from $\mathcal{Z}$
3:     $a = \pi_\phi(\mathbf{obs})$
4:     $\hat{z} = f_\theta(a)$
5:     $z' = \arg\min_{z \in \mathcal{Z}'} \|z - \hat{z}\|_2$
6:     $\hat{l} = g(z')$
7:     Execute $\hat{l}$
8: end procedure
```

3.4.3 Observation and Action space

The observation space for the RL agent is as defined in Table 3.1. The observation space is similar to that in chapter 2 with changes to remove the use of the curriculum for training. The observations can be divided into four categories: 1) ego-vehicle state; 2) goal and reference path-related observations; 3) surrounding vehicles-related observations; and 4) road network-related observations. The dynamics constraints are based on the kinematic bicycle model (`KS2`) as described in [5]. The reference path observation consists of waypoints from the path generated by A-star search over the lanelet network in Commonroad. The distance advancements for longitudinal and lateral directions are calculated along this reference path. For the surrounding vehicles, we include the current state of each of the five vehicles in the ego-vehicle's frame.

The action space for the PPO agent is a continuous action space of $\mathbb{R}^{30}$, which represents the size of a primitive vector.

3.4.4 Reward Structure

The components of the reward function for PPO are as follows:

- A positive reward for distance advancement along the reference path as described in section 3.4.3. The reward is calculated as $\rho_1 d_{long} + \rho_2 d_{lat}$ where ρ_1 and ρ_2 are positive constants.
- A constant positive reward ρ_3 for reaching the goal within the allowed orientation error.

Table 3.1: Observation Space for the RL agent

Variable	**Description**	**Values**
v_t	Ego-vehicle absolute velocity	$\mathbb{R}$
a_{t-1}	Ego-vehicle previous acceleration	$\mathbb{R}$
δ_{t-1}	Ego-vehicle previous steering	$\mathbb{R}$
θ_t	Ego-vehicle orientation	$\mathbb{R}$
$\dot{\theta}_t$	Ego-vehicle turn-rate	$\mathbb{R}$
r	Reference path waypoints (x, y, θ) in ego-vehicle frame	$\mathbb{R}^{15}$
$d\theta_t$	Ego-vehicle orientation deviation from reference path	$\mathbb{R}$
d_{long}	Longitudinal distance advancement towards goal	$\mathbb{R}$
d_{lat}	Latitudinal distance advancement towards goal	$\mathbb{R}$
t_{out}	Time remaining before time-out	$\mathbb{R}$
v_{ref}	Velocity deviation from reference	$\mathbb{R}$
c	Lane curvature	$\mathbb{R}$
l_o	Ego-vehicle offset from lane centerline	$\mathbb{R}$
r_l	Ego-vehicle distance from left road boundary	$\mathbb{R}$
r_r	Ego-vehicle distance from right road boundary	$\mathbb{R}$
l_l	Ego-vehicle distance from left lane boundary	$\mathbb{R}$
l_r	Ego-vehicle distance from right lane boundary	$\mathbb{R}$
o	Target vehicles' states(x, y, v, θ)	$\mathbb{R}^{20}$

- A constant negative reward σ_1 for collision with obstacles and going off-road.
- A constant negative reward σ_2 for time-out.
- A negative reward for high acceleration ($\sigma_3 a_t^2$), steering-angle ($\sigma_4 \delta_t^2$) and steering-angle turn rate ($\sigma_5 \dot{\delta_t}^2$).

An episode is terminated when any one of the following conditions is met: ego-vehicle reaches the goal within allowed orientation error, ego-vehicle goes off-road, ego-vehicle collides with another obstacle or time-out.

3.5 Experiments

We test our method on the CommonRoad simulator for unsignalized T-intersections and four-way intersections planning and compare it with the CommonRoad leader-board and nominal motion primitive lattice planner available in CommonRoad. We use the kinematic single-track vehicle BMW320i for our experiments. We also perform an ablation study to compare our method against using a classifier-based reinforcement learning agent.

3.5.1 Experimental Setup

We use the same experimental setup for scenarios as described in section 2.5.1.

3.5.2 Motion Primitives Library

We used the CommonRoad-Search package in the simulator to generate the motion primitives for a fair comparison with their search planner. However, the method can be used with any other motion primitive library. In Figure 3.2, we show some motion primitives for a particular state of the autonomous vehicle.

The motion primitives are generated by sampling different combinations of initial and target states. They have varying initial and final velocities ranging from 0 - 10 m/s and varying steering angles between -1.0 and 1.0 radians. Each primitive has a duration of 0.5 seconds with a discretization of 0.1 seconds. The control inputs are constant during this period. The motion primitives are generated for all combinations of the velocities and steering angles at the initial state and the final state. We used a

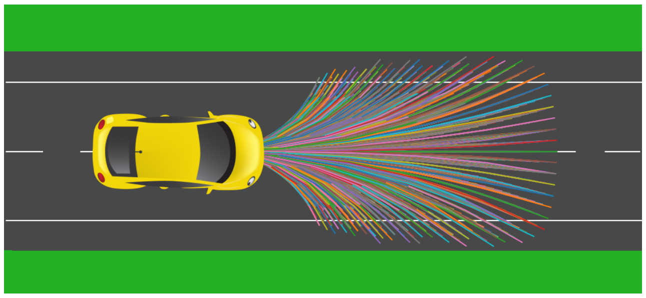

Figure 3.2: Visualization of sample motion primitives.

discretization of 0.25 m/s for the velocity samples and 0.1 radians for steering angles. This makes a total of 41 possible velocities and 21 steering angles for the initial and final states. Thus the total number of initial and final states considered is $(41 \times 21 = 861)$ each. Hence, the total number of primitives considered is $(861 \times 861 = 741321)$. Since all possible final states are not reachable from every initial state, the final number of primitives after pruning equals 58810. Two motion primitives are considered connectable if the velocity and the steering angle of the final state of the preceding primitive are equal to those of the initial state of the following primitive within a given threshold.

Each primitive is defined by initial state, final state and four intermediate waypoints $l \in \mathbb{R}^{30}$. Each state/waypoint consists of position, velocity, orientation and steering angle.

3.5.3 Network Architecture

We used a two-layer encoder with ReLU activation to compress the primitives from $\mathbb{R}^{30}$ to $\mathbb{R}^{10}$. The decoder is similar to an encoder with the opposite configuration.

To represent the PPO policy, we used a fully-connected (FC) network with 8 hidden layers of 64 units each and ReLU nonlinearities. The output layer consists of 30 units and is used as an input to the encoder to generate the optimal primitive embedding. The hyperparameter values for PPO are listed in table 3.2.

Table 3.2: Hyperparameters for training the RL agent

Parameter	**Value**
Training Iterations	1200
Discount (γ)	0.99
GAE parameter (λ)	0.95
Clipping parameter	0.2
Learning-rate	0.0005
No. of parallel environments	32
Time horizon	512
Batch size	16384

3.5.4 Training Details

All training and benchmarking was done on a desktop with a 3.5 GHz AMD Ryzen 9 5900hs CPU. We used a customized version of the CommonRoad-RL framework for training and testing.

3.5.5 CommonRoad Benchmarks

We evaluate our method on the CommonRoad Benchmarks leaderboard and also compare it with the CommonRoad search planner for a more thorough comparison, as we do not have access to the planners from other participants. The default planner is designed for solving entire scenarios together, which is not feasible for the autonomous driving use case, as the planning horizon is very long. We implement a closed-loop version of the search-based planner for comparison by integrating the CommonRoad route planner, which generates intermediate goals to follow. The waypoints for the PPO agent are also computed using the same route planner. The search method used is A-star.

3.5.6 Performance Evaluation

The metrics used for comparison are trajectory computation time, success rate and trajectory cost. For T-intersections, we also consider the ranking of the method in the leaderboard. The ranking is decided for each test scenario independently.

Our method uses 58810 primitives in all comparisons. We compare against multiple different configurations of primitives in the search planner, which are as follows:

1. SP-A: The default configuration of our method with 58810 primitives.
2. SP-B: 20814 primitives with velocity discretization of 0.5 m/s and steering discretization of 0.1 radians.
3. SP-C: 25509 primitives with velocity discretization of 0.25 m/s and steering discretization of 0.2 radians.

The results presented are for 164 T-intersection testing scenarios and 600 Four-way intersection scenarios. For leaderboard, the metrics are computed for the best solution before our method is submitted to each of the planning scenarios.

Computation Time

Table 3.3 shows the average trajectory computation time comparison for completing the scenarios.

Table 3.3: Average computation time in seconds

Scenario	T-intersections	Four-way intersections
Our method	0.96	0.98
Leaderboard	8.01	-
SP-A	3.53	3.92
SP-B	1.76	2.55
SP-C	2.23	2.85

Success Rate

A test problem is considered successful when the ego-vehicle reaches the goal region within the required orientation, velocity and time-interval without any collision or going off-road. Table 3.4 shows the performance of the methods.

For the ablation study, we use a PPO agent as a nominal classifier with 512 feasible primitives generated in a manner similar to our method, but with a coarser resolution. A lower number of primitives is used, as the classifier is unable to handle a larger primitive library. For the PPO agent, we use a fully-connected (FC) network with 2 hidden layers of 1024 units each and ReLU nonlinearities. The output head consists of 512 units with a softmax nonlinearity.

Table 3.4: Success Rate Comparison

Scenario	**T-intersections(%)**	**Four-way intersections (%)**
Our method	99.39	90.16
Leaderboard	100.00	-
SP-A	97.56	92.83
SP-B	93.29	87.00
SP-C	91.46	85.66
PPO Classifier	45.12	37.81

The direct classifier performs poorly in comparison to all the methods. This is because of the large number of possible primitives under consideration, establishing the need for our method.

Cost Evaluation

We use the `SM1` cost evaluation function of CommonRoad for comparison. The cost is calculated as a weighted sum of acceleration, steering angle, steering turn rate, offset from lane center, velocity offset and distance from obstacles. Table 3.5 shows the average cost for the methods.

Table 3.5: Trajectory Cost Comparison

Scenario	**T-intersections**	**Four-way intersections**
Our method	16826.37	4392.65
Leaderboard	17770.78	-
SP-A	16974.61	4223.31
SP-B	23177.22	5440.01
SP-C	25718.78	4480.26

In addition to having the lowest average cost, for 108 out of 164 T-intersection test scenarios, our solution is the best-performing (least cost) among all submissions on the leaderboard.

The major drawback of the search-based method is the limited search depth of the planner. Fig. 3.3 shows a failure case of the planner. Due to the planner's limited depth, the planner decides to drive (path in blue) towards an incoming vehicle at a high-velocity. This drives the vehicle to a state from which the primitives are unable to recover it. However, the learnt agent is able to do long-term planning better, thus leading to a higher success rate.

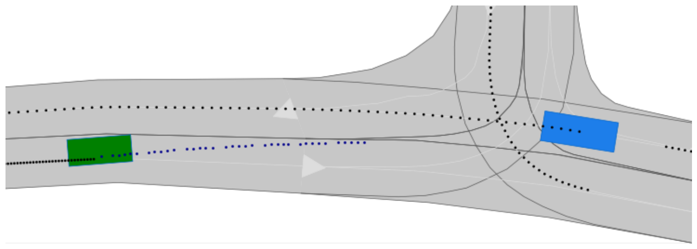
Figure 3.3: **Search Planner failure case.** This is a result of the limited search-depth of the conventional planner.

3.5.7 Discussion

The results above show that the biggest advantage of our method over using the traditional search planner is the reduction in computation time. This is because of the RL agent's being able to learn the appropriate primitive to execute. Our method is almost 8x faster than the best leaderboard solution and 4x faster than the closed loop planner. The success rate and the trajectory cost results also show that this reduction in computation time does not affect the performance significantly. For T-intersections, our method has comparable performance with the Leaderboard and better performance than the closed-loop search planner in all configurations. The overall cost is also much lower compared to other methods. Though the success rate of our solution in four-way intersections is lower than that of SP-A, the high computation time of SP-A makes it unfit for real-world use cases. The trajectory cost of our method is comparable to SP-A. The comparisons also confirm that decreasing the number of primitives to make the search-based approach feasible in real-time significantly degrades the performance. Our method has the advantage of considering a large number of primitives as compared to traditional search methods without affecting the run-time significantly.

3.6 Conclusion

In this work, we propose a reinforcement learning framework for motion primitive-based planning. It demonstrates how reinforcement learning can be used with large motion primitive libraries to generate primitives-based trajectories for self-driving much faster than traditional search without compromising the performance of the planner. The framework, unlike end-to-end frameworks, can also be used for safety-critical systems, as the trajectory generated can be further optimized or checked for collisions before execution. We test the performance of our framework on the highly dynamic intersection traversal task for autonomous driving in the CommonRoad simulator. The framework outperforms the CommonRoad leaderboard benchmarks in the T-intersection scenarios and performs comparably to the best search-based planner in the four-way intersection scenarios. Moreover, its computation time is much lower for both the scenarios, making it fit for real-world applications.

Chapter 4

Offline Reinforcement Learning with Latent Diffusion

In certain safety-critical systems, any form of online data collection for RL training might not be possible. Offline reinforcement learning (RL) 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*. Several methods [19, 35, 37] have been proposed for offline RL, all of which aim to strike a balance between constraining the learned policy to the support of the behavior policy and improving upon it. At the core of many of these approaches is an attempt to mitigate the *extrapolation error* which arises while querying the learned Q-function on out-of-support samples for policy improvement. For example, 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 [19].

Batch-Constrained deep Q-Learning (BCQ) [19] was proposed to model the data distribution using a state-conditioned generative model which can propose candidate actions conditioned on the states. During evaluation, the candidate state-actions pairs are scored using a learnt Q-function on the dataset. Since the generative model is trained on the trajectory dataset, the candidates are expected to be in-support where the Q-function is accurate, avoiding extrapolation error. However, this relies on the assumption that sampling from the generative model does not sample out-of-distribution samples.

Another challenge in offline RL arises while stitching portions of demonstrations to improve over the behavior policy. Such stitching can be much more easily achieved over high-level behaviors instead of low-level policies, as there is a clearer distinction among behaviors at high-level and credit-assignment is easier. We demonstrate this further with our experiments.

Thus, a straightforward approach to offline RL would be doing Batch-Constrained Q-Learning over learnt high-level latent spaces. However, as we show in section 4.4.2, such high-level behaviors are multi-modal and proposed VAEs are unable to learn good state-conditioned priors for such spaces.

In this work, we present a novel method for achieving batch-constraining in latent space for offline RL. Our method uses Latent Diffusion Models [53], which are currently state-of-the-art generative models for image-generation [52, 55], to achieve modeling complex latent distributions and sample from the diffusion model to generate candidates for evaluation using a learnt Q-function which in turn is also learnt using samples from the diffusion model. Learning the Q-function using in-support samples generated from the diffusion model ensures that extrapolation or bootstrapping error is mitigated. Further, our method also allows for learning temporally extended world models, making our method more interpretable. 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.

4.1 Related Work

4.1.1 Offline reinforcement learning

As discussed previously, offline RL poses the challenge of distributional shift while stitching suboptimal trajectories together. Conservative Q-Learning (CQL) [37] 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) [70] 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 [68], speech synthesis [34] and image-generation [52, 55], [11] proposed to use a generative model for offline RL and bypass the need for Q-learning or bootstrapping altogether with *return-conditioning* [36, 65]. 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's 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 [19]. 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 stitching, [49] and [3] 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. We show that as we increase the horizon for temporal abstraction, the corresponding latent spaces incorporate rich multimodal behavioral representations that can facilitate simpler credit assignment and skill stitching.

4.1.2 Diffusion Probabilistic Models

Recently, diffusion models [62, 63] have emerged as state-of-the-art generative models for conditional image-generation [52, 55], super-resolution [56] and inpainting [42]. They are a much more powerful class of generative model compared to Variational Autoencoders (VAEs) [33], and benefit from a more stable training process as compared to Generative Adversarial Networks (GANs) [21]. Recent works in offline RL [31], [2] have proposed using diffusion to model trajectories and showcased its effectiveness in stitching together behaviors to improve upon suboptimal demonstrations. However, [31] makes 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, [2] 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. The idea is to diffuse over semantically rich latent representations while relying on powerful decoders for high-frequency details.

In summary, rather than avoiding Q-learning, we model the behavioral policy with diffusion and use this to avoid extrapolation error through batch-constraining. We also harness the expressivity of powerful diffusion generative models to reason with temporal abstraction and improve credit assignment. Further, prior works which explored diffusion for offline RL [2, 31] directly diffused over the raw state-action space, and their architectural considerations for effective diffusion models limited the networks to be simple U-Nets [54]. The separation of the diffusion model from the low-level policy allows us to model the low-level policy using a powerful autoregressive decoder.

4.2 Background

4.2.1 Diffusion Probabilistic Models

Diffusion models [62, 63] 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*:

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1}), \qquad q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}) \tag{4.1}$$

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* $p_\psi(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$:

$$p_\psi(\mathbf{x}_{0:T}) := p(\mathbf{x}_T) \prod_{t=1}^{T} p_\psi(\mathbf{x}_{t-1} \mid \mathbf{x}_t), \qquad p_\psi(\mathbf{x}_{t-1} \mid \mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \mu_\psi(\mathbf{x}_t, t), \mathbf{\Sigma}_\psi(\mathbf{x}_t, t)) \tag{4.2}$$

The reverse process is trained by minimizing a surrogate loss-function [29]:

$$\mathcal{L}(\psi) = \mathbb{E}_{t \sim [1,T], \mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0},\mathbf{I})} \mid\mid \epsilon - \epsilon_\psi(\mathbf{x}_t, t) \mid\mid^2 \tag{4.3}$$

Diffusion can be performed in a compressed latent space $\mathbf{z}$ [53] 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 β-Variational Autoencoder [27, 33] with a low β. The prior is then trained to fit the optimized latents of this model. We explain this two-stage training in more detail in section 4.3.1.

4.2.2 Offline Reinforcement Learning

Offline RL [41] is an extension of the RL problem described in section 1.2. In this setting, 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 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 the extrapolation error problem.

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 additional methods to use the latent diffusion prior with goal-conditioning and model-based planning, which are more suitable for certain navigation tasks.

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 β-Variational Autoencoder (VAE) [27, 33]. Specifically, we maximize the evidence lower bound (ELBO):

$$\mathcal{L}(\theta, \phi, \omega) = \mathbb{E}_{\boldsymbol{\tau}_H \sim D}[\mathbb{E}_{q_\phi(\mathbf{z}|\boldsymbol{\tau}_H)}[\sum_{t=0}^{H-1} \log \pi_\theta(\mathbf{a}_t \,|\, \mathbf{s}_t, \mathbf{z})] - \beta D_{KL}(q_\phi(\mathbf{z} \,|\, \boldsymbol{\tau}_H) \;||\; p_\omega(\mathbf{z} \,|\, \mathbf{s}_0))] \tag{4.4}$$

where q_ϕ represents our approximate posterior over $\mathbf{z}$ given $\boldsymbol{\tau}_H$, and p_ω 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 β-VAE to learn a latent space to diffuse over. As such, the prior p_ω 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. Prior works [3, 49] 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 β 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 π_θ is represented as an autoregressive model which can capture the fine details across the action dimensions, and is similar to the decoders used by [20] and [3]. 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, $\mathbf{z} \sim q_\phi(\mathbf{z} \mid \boldsymbol{\tau}_H)$ where q_ϕ is the VAE encoder trained earlier, and $\mathbf{s}_0$ is the first state in $\boldsymbol{\tau}_H$. We want to model the prior $p(\mathbf{z} \mid \mathbf{s}_0)$, 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 $p_\psi(\mathbf{z} \mid \mathbf{s}_0)$ 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 [52] and [32], 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 the Min-SNR-γ strategy [24]. 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. 4.3 to minimize:

$$\mathcal{L}(\psi) = \mathbb{E}_{t\sim[1,T],\boldsymbol{\tau}_H\sim\mathcal{D},\mathbf{z}_0\sim q_\phi(\mathbf{z}|\boldsymbol{\tau}_H),\mathbf{z}_t\sim q(\mathbf{z}_t|\mathbf{z}_0)}[\min\{\text{SNR}(t), \gamma\}(||\ \mathbf{z}_0 - \mu_\psi(\mathbf{z}_t, \mathbf{s}_0, t)\ ||^2)] \tag{4.5}$$

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 [29] to sample from the diffusion prior in this work due to its simple implementation. As proposed in [28], we use classifier-free guidance for diffusion. 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.

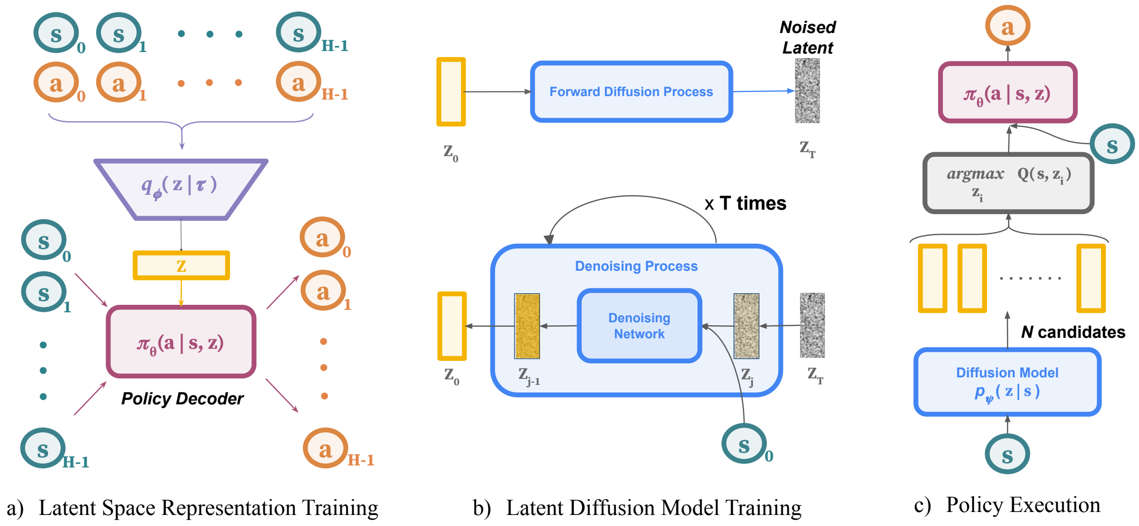

Figure 4.1: **Latent Diffusion Reinforcement Learning Overview** a) We first learn the latent space and low-level policy decoder by training a β-VAE over H-length sequences from the demonstrator dataset. b) We train a latent diffusion prior conditioned on $\mathbf{s}_0$ to predict latents generated by the VAE encoder. c) After learning a Q function using LDCQ (Algorithm 4), we score latents sampled by the prior with this Q function and execute the low-level policy π_θ conditioned on the argmax latent.

4.3.2 Latent Diffusion-Constrained Q-Learning (LDCQ)

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 [19].

$$\pi(\mathbf{s}) = \underset{\substack{\mathbf{a} \\ s.t.(\mathbf{s},\mathbf{a})\in\mathcal{D}}}{\operatorname{argmax}} Q(\mathbf{s}, \mathbf{a}) \tag{4.6}$$

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:

$$\pi(\mathbf{s}) = \underset{\mathbf{a}_i \sim \pi_\psi(\mathbf{a}|\mathbf{s})}{\operatorname{argmax}} Q(\mathbf{s}, \mathbf{a}_i) \tag{4.7}$$

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.

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 γ-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 the H-length trajectory snippet. The Q-function is learned with temporal-difference updates [67], where we sample a batch of latents for the target argmax using the diffusion prior $p_\psi(\mathbf{z} \,|\, \mathbf{s}_{t+H})$. This should only sample latents which are under the support of the behavioral policy, and hence with the right temporal abstraction, this allows for stitching skills to learn an optimal policy grounded on the data support. The resulting Q update can be summarized as:

$$Q(\mathbf{s}_t, \mathbf{z}) \leftarrow (r_{t:t+H} + \gamma^H Q(\mathbf{s}_{t+H}, \underset{\mathbf{z}_i \sim p_\psi(\mathbf{z}|\mathbf{s}_{t+H})}{\operatorname{argmax}} (Q(\mathbf{s}_{t+H}, \mathbf{z}_i)))) \tag{4.8}$$

We use Clipped Double Q-learning as proposed in [18] to further reduce overestimation bias during training. We also use Prioritized Experience Replay [57] to accelerate the training in sparse-reward tasks like AntMaze and FrankaKitchen. We summarize our proposed LDCQ method in Algorithm 4.

Algorithm 4 Latent Diffusion-Constrained Q-Learning (LDCQ)

```
1: 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 $\alpha_1, \ldots, \alpha_T$, $\bar{\alpha}_1, \ldots, \bar{\alpha}_T$,
   $\beta_1, \ldots, \beta_T$.
2: 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 $\Theta_1^{target} \leftarrow \Theta_1$, $\Theta_2^{target} \leftarrow \Theta_2$
3: for $iter = 1$ to $M$ do
4:     Sample a minibatch of $N$ transitions $\{(\mathbf{s}_t, \mathbf{z}, r_{t:t+H}, \mathbf{s}_{t+H})\}$ from $\mathcal{B}$
5:     Sample $n$ latents for each transition: $\mathbf{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
6:     for $t = T$ to 1 do                                        ▷ DDPM Sampling
7:         $\hat{\mathbf{z}} = \mu_\psi(\mathbf{z}_t, \varnothing, t) + w(\mu_\psi(\mathbf{z}_t, \mathbf{s}_{t+H}, t) - \mu_\psi(\mathbf{z}_t, \varnothing, t))$
8:         $\mathbf{z}_{t-1} \sim \mathcal{N}(\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{z}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\hat{\mathbf{z}}, \mathbb{I}(t > 1)\beta_t \mathbf{I})$
9:     end for
10:    Compute the target values $y = r_{t:t+H} + \gamma^H \{\max_{\mathbf{z}_0}\{\min_{j=1,2} Q_{\Theta_j^{target}}(\mathbf{s}_{t+H}, \mathbf{z}_0)\}\}$
11:    Update $Q$-networks by minimizing the loss: $\frac{1}{N}||y - Q_\Theta(\mathbf{s}_t, \mathbf{z})||_2^2$
12:    Update target $Q$-networks: $\Theta^{target} \leftarrow \rho\Theta + (1-\rho)\Theta^{target}$
13: end for
```

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 [31] 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 $p_\psi(\mathbf{z} \mid \mathbf{s}_0, \mathbf{s}_g)$, 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)

We explore another method to derive a policy for offline RL with latent diffusion with a model-based approach which learns a temporally abstract world model of the environment from 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 $\mathbf{s}_{t+H}$. 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:

$$\mathcal{L}(\eta) = \mathbb{E}_{\boldsymbol{\tau}_H \sim \mathcal{D}} \mid\mid p_\eta(\mathbf{s}_{t+H} \mid \mathbf{s}_t, \mathbf{z}) - \mathbf{s}_{t+H} \mid\mid^2 \tag{4.9}$$

where η are the parameters of the temporally abstract world model p_η.

In goal-reaching environments, we leverage this model to do planning using the diffusion prior. We sample n latents $\mathbf{z}^i$ $(1 \leq i \leq n)$ using the diffusion prior for the current state $\mathbf{s}_t$, and use the learnt dynamics model to compute predicted future state $\mathbf{s}^i_{t+H}$ for each latent $\mathbf{z}^i$. These final states are then scored using a cost-function $\mathcal{J}$ and the latent corresponding to the best final state is chosen for execution. Note that sampling latents from the diffusion prior ensures that the world model is not queried on out-of-support data. We refer to this method as *Latent Diffusion-Constrained Planning (LDCP)*. The planning procedure is summarized in Algorithm 5.

Algorithm 5 Latent Diffusion-Constrained Planning (LDCP)

```
1: 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 $\alpha_1, \ldots, \alpha_T$, $\bar{\alpha}_1, \ldots, \bar{\alpha}_T$, $\beta_1, \ldots, \beta_T$.
2: $done = False$
3: while not $done$ do
4:     Observe environment state $\mathbf{s}_0$
5:     Sample $n$ latents: $\mathbf{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
6:     for $t = T$ to 1 do                                   ▷ DDPM Sampling
7:         $\hat{\mathbf{z}} = \mu_\psi(\mathbf{z}_t, \varnothing, t) + w(\mu_\psi(\mathbf{z}_t, \mathbf{s}_0, t) - \mu_\psi(\mathbf{z}_t, \varnothing, t))$
8:         $\mathbf{z}_{t-1} \sim \mathcal{N}(\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{z}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\hat{\mathbf{z}}, \mathbb{I}(t > 1)\beta_t \mathbf{I})$
9:     end for
10:    Compute future states for each latent $\mathbf{z}_0^i$: $\mathbf{s}_H^i = p_\eta(\mathbf{s}_H^i \mid \mathbf{s}_0, \mathbf{z}_0^i)$
11:    Find best latent based on the cost-function: $i = \underset{i}{\operatorname{argmin}}\ \mathcal{J}(\mathbf{s}_H^i)$
12:    Compute action-sequence using policy decoder $\pi_\theta(\mathbf{a} \mid \mathbf{s}_0, \mathbf{z}_0^i)$
13:    $h = 0$
14:    while $h < H$and not $done$ do
15:        Execute action $\mathbf{a}_h$
16:        Update $done$
17:        $h = h + 1$
18:    end while
19: end while
```

The cost-function which we use for the goal-reaching environments is the Euclidean distance to the goal. We can also extend this planning to horizons greater than H by further sampling latents for each future state $\mathbf{s}_{t+H}^i$ $(1 \leq i \leq n)$. This means, after sampling n latents for $\mathbf{s}_t$ with the diffusion prior, we further sample n more latents for each of the future states $\mathbf{s}_{t+H}^i$. This increases the 'planning depth' d. The final states at the last level of planning are then scored using the cost-function and the latent at the first level which led to that final state is chosen for execution. This procedure complexity grows exponentially and thus the planning depth has to be restricted.

4.3.5 Visualizing Model Predictions

Learning a world model also allows us to visualize the effect of executing any given latent behavior. This means, even when the model is not used for planning, like in LDCQ, it can be used to compute the final state that will be reached for every latent behavior from a particular state. This information can be used to understand if the model is learning reasonable behavior modes. During testing as well, this can be used for safety checks.

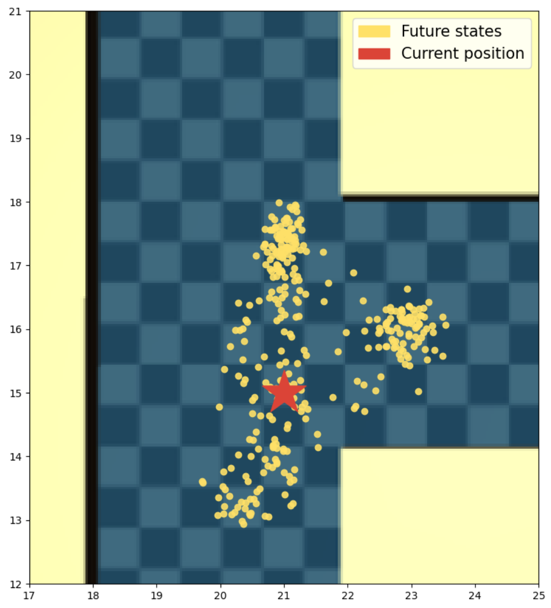

Figure 4.2: **Visualizing model predictions:** Visualization of future states with latents sampled from the diffusion prior at a T-intersection in antmaze-large-diverse-v2 D4RL task.

We plot the xy-coordinates of our abstract world model $p_\eta(\mathbf{s}_{t+H} \mid \mathbf{s}_t, \mathbf{z})$ predictions at a T-intersection in the AntMaze large environment for latents sampled from our diffusion prior $\mathbf{z} \sim p_\psi(\mathbf{z} \mid \mathbf{s}_t)$ in Figure 4.2 to demonstrate this. The plot shows that the diffusion prior sampled latents which go in all the three directions at the T-intersection.

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 4.4.5).

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 [17]. 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 [17] 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 4.3, 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 4.3). 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.

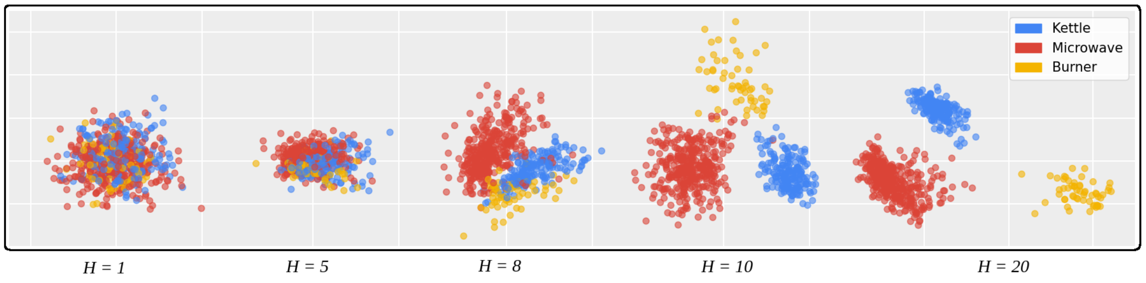

Figure 4.3: **Projection of latents across horizon**. Latent projections of trajectory snippets with different horizon lengths H. From the initial state there are 3 tasks (Kettle, Microwave, Burner) which are randomly selected at the start of each episode. These 3 primary modes emerge as we increase 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 4.4 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 (Figure 4.5) and the VAE conditional prior (Figure 4.6) 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.

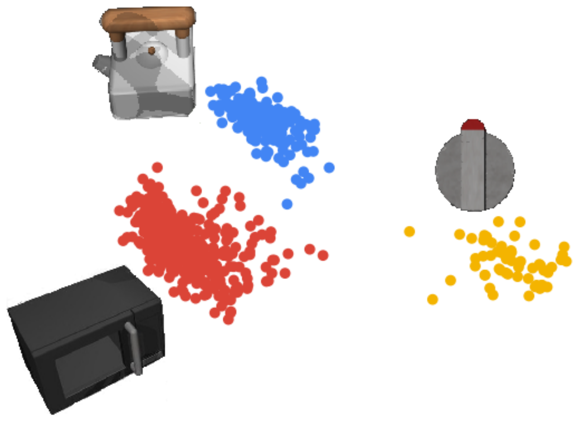

Figure 4.4: Visualization of projection of Ground truth latents

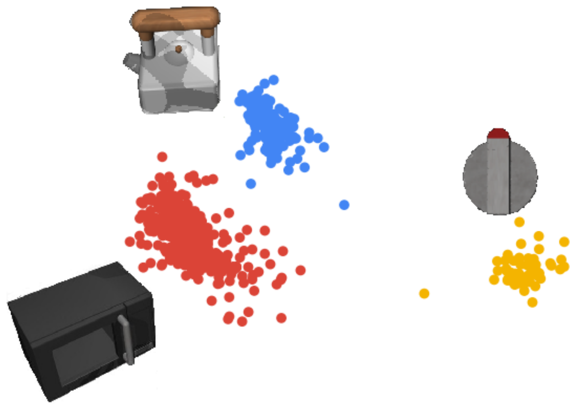

Figure 4.5: Visualization of projection of latents from the diffusion prior

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

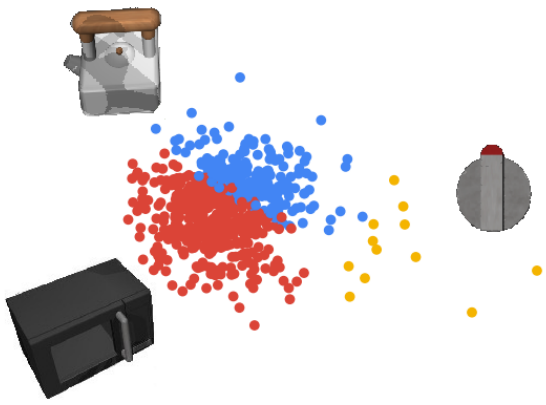

Figure 4.6: Visualization of projection of latents from the VAE prior

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 4.7.

4.4.4 Network Architecture

Variational Autoencoder

Encoder. For learning the latent trajectory representation, our VAE uses an architecture similar to [3]. The encoder consists of two stacked bidirectional GRU layers, followed by mean and standard deviation heads which are each a 2 layer MLP with RELU activation for the hidden layers. The mean output head is a linear layer. The standard deviation output head is followed by a SoftPlus activation function to ensure it is always positive. The hidden layer dimension is fixed to 256.

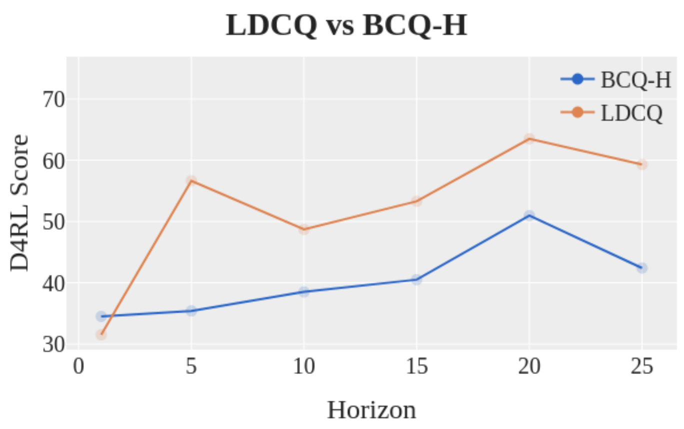

Figure 4.7: D4RL score of LDCQ and BCQ-H on kitchen-mixed-v0 with varying sequence horizon H

Decoder. For the low-level policy decoder, we use an auto-regressive policy network similar to that described in EMAQ [20], in which each element of the action vector has its own MLP network, taking as input the current state, latent representation, and all previously-sampled action elements. The complete action vector is sampled element-by-element, with the most recently sampled element becoming an input to the network for the next element. These MLP networks consists of 2 layers followed by 2 layers of mean and standard deviation heads similar to the encoder network. The mean output head is a linear layer and the standard deviation output head is followed by a SoftPlus activation. Again, ReLU activation is used after all hidden layer and the hidden dimension is fixed to 256.

Diffusion Prior

The diffusion prior is a deep ResNet [25] architecture consisting of 8 residual blocks. It takes as input a vector representing a latent trajectory $\mathbf{z}$ and outputs a denoised version of the latent. The hidden blocks are of dimensions: [128, 32, 16, 8, 16, 32, 128]. Similar to a U-Net [54], the initial blocks are connected by residual connections to the later blocks having the same hidden dimension. The diffusion timestep t is encoded with a 256-dimensional sinusoidal embedding and then further encoded with a 2-layer MLP. The conditioning state $\mathbf{s}$ is also encoded by a 2 layer MLP. In each residual block, the state and time encodings are concatenated with the current layer activation for conditioning. When training the unconditional diffusion model for classifier-free guidance, the state input is given as a vector of zeros to represent a null vector.

Q-networks

The Q-networks take as input the state $\mathbf{s}$, latent $\mathbf{z}$ and consist of a 5 layer MLP with 256 hidden units in the first 3 layers, 32 hidden units in the third layer, and finally a linear output layer. We use GELU activation function between hidden layers. LayerNorm [7] is applied before each activation.

4.4.5 Offline RL benchmarks

In this section, we investigate the effectiveness of our Latent Diffusion Reinforcement Learning methods on the D4RL offline RL benchmark suite [17]. We compare with Behavior Cloning and several *state-of-the-art* offline RL methods: Batch Constrained Q-Learning (BCQ) [19], Conservative Q-Learning (CQL) [37], Implicit Q-Learning (IQL) [35], Decision Transformer (DT) [11], Diffuser [31] and Decision Diffuser [2]. The last two algorithms are previous trajectory diffusion methods.

Hyperparameters

We found that our method does not require much hyperparameter tuning and only had to vary the sequence horizon H across tasks. In maze2d, AntMaze and Carla 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 Tables 4.1, 4.2 and 4.3.

Hardware

The models were trained on NVIDIA RTX A6000. Since different tasks have different dataset sizes, the model training times changes across tasks. Depending on the task, training the β-VAE took between 3-7 hours, the diffusion prior took between 4-12 hours and the Q-Learning took between 3-5 hours.

Table 4.1: β-VAE hyperparameters

Parameter	**Value**
Learning rate	5e-5
Batch size	128
Epochs	100
Latent dimension ($\mathbf{z}$)	16
β	0.05
Hidden layer dimension	256

Table 4.2: Diffusion training hyperparameters

Parameter	**Value**
Learning rate	1e-4
Batch size	32
Epochs	300
Diffusion steps (T)	500
Drop probability (For unconditional prior)	0.1
Variance schedule	linear
Sampling algorithm	DDPM
γ (For Min-SNR-γ weighing)	5

Table 4.3: Q-Learning hyperparameters

Parameter	**Value**
Learning rate	5e-4
Batch size	128
Discount factor (γ)	0.995
Target net update rate (ρ)	0.995
PER buffer α	0.7
PER buffer β	Linearly increased from 0.3 to 1, Grows by 0.03 every 3000 steps
Diffusion samples for batch argmax	500

Results

In Table 4.4, 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 difficult 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.

Table 4.4: Performance comparison on D4RL tasks which require long-horizon stitching with high multimodality. Goal conditioning (LDGC) and Planning (LDCP) variant 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** ± 3.8
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** ± 1.1
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** ± 1.3
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	-	-

All our methods (LDCQ, LDGC, LDCP) achieve state-of-the-art results in all sparse-reward D4RL tasks. The goal-conditioned and planning variants outperform all others in maze2d and AntMaze. These variants are do not require Q-learning and are ideal for goal-reaching tasks.

We visualize the performance of our method (Fig. 4.8) compared to BCQ (Fig. 4.9) in the kitchen-mixed-v0 task. We see that while LDCQ is able to do the tasks sequentially and complete 3 tasks successfully, the BCQ agent after completing the kettle task gets stuck in the middle of switch and burner. This is a result of averaging between different tasks as we also see in section 4.4.2.

We also provide an evaluation of our method on the D4RL locomotion suite (Table 4.5). While these tasks are not specifically focused on trajectory-stitching,

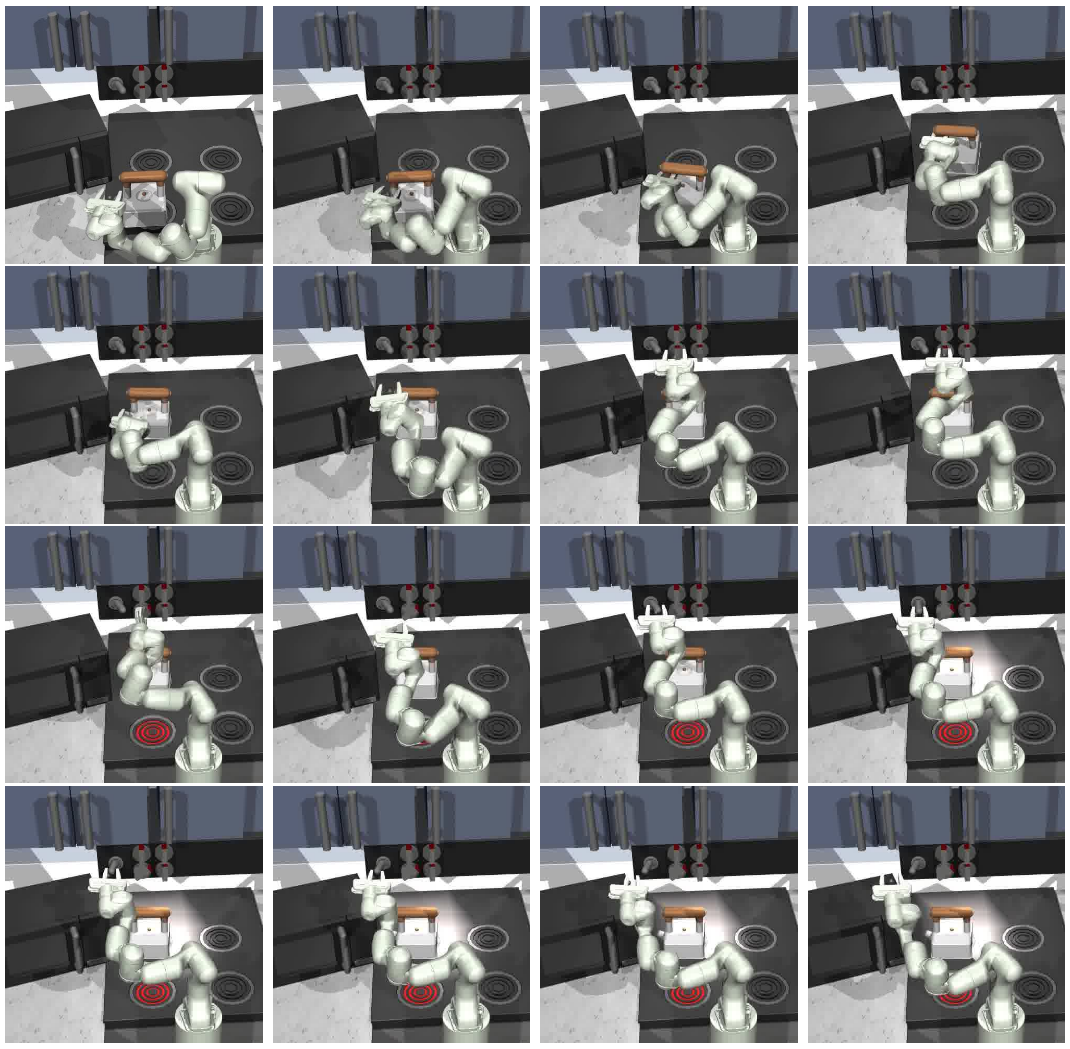

Figure 4.8: **LDCQ in kitchen-mixed-v0 environment.** The sequence shows a single episode where the agent completes the kettle task first followed by burner and switch.

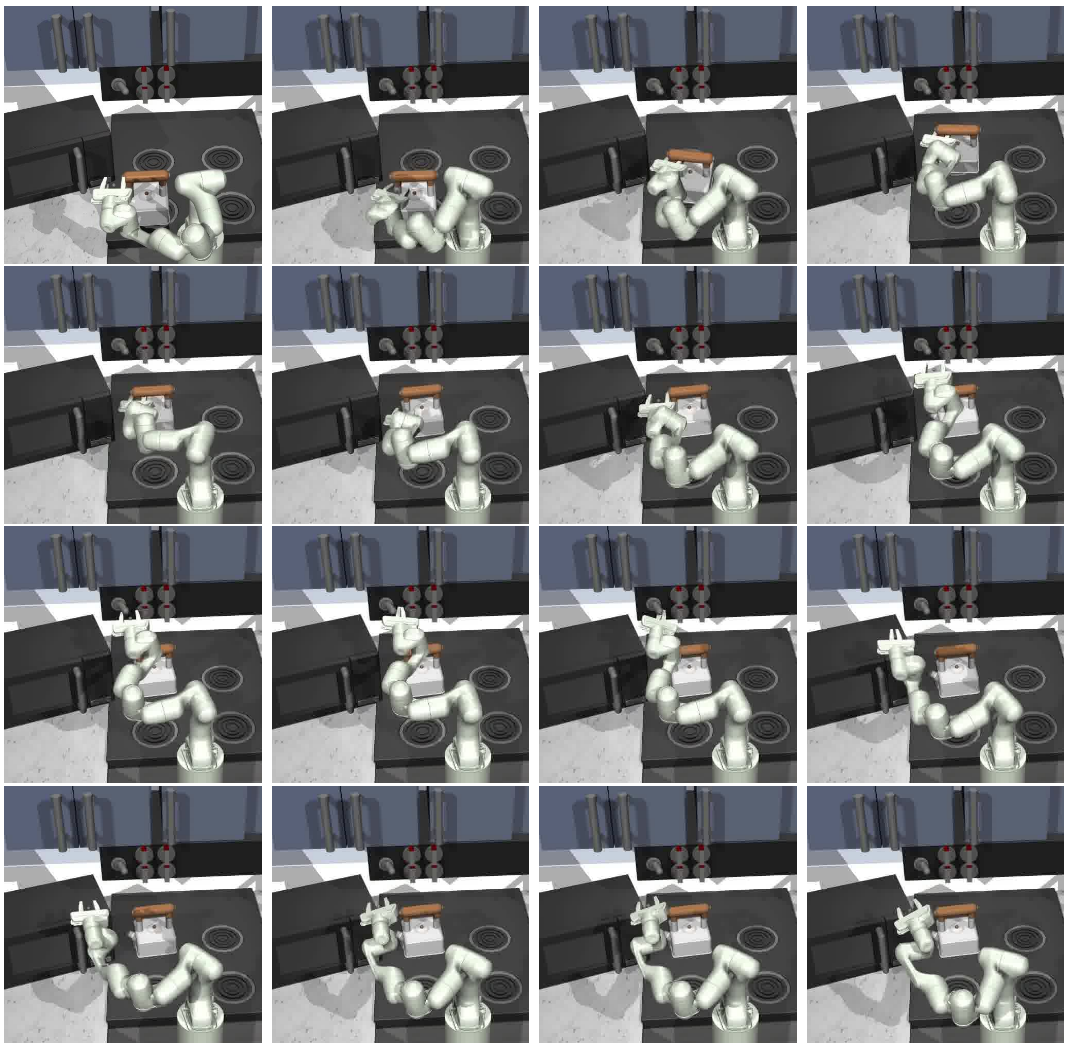

Figure 4.9: **BCQ in kitchen-mixed-v0 environment.** The sequence shows a single episode where the agent completes the kettle task and then tries to complete burner and switch. However the arm instead goes in the middle of the burner and switch because the VAE is unable to segregate the two tasks and chooses actions by averaging over the two tasks.

our method is competitive with other offline RL methods. We only run the LDCQ variant here since they are not goal-reaching tasks.

Table 4.5: 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

To extend our method for tasks with high-dimensional image input spaces, we propose to compress the image space using an autoencoder such that our method operates on a compressed state space. This essentially means we create a low-dimensional compressed representation using an encoder $\mathcal{E}$ before using the LDCQ framework. Note that this encoder operates on a single image and not on a temporal sequence of images (Figure 4.10). The downstream framework of LDCQ, however, operates on the temporal compressed image sequences.

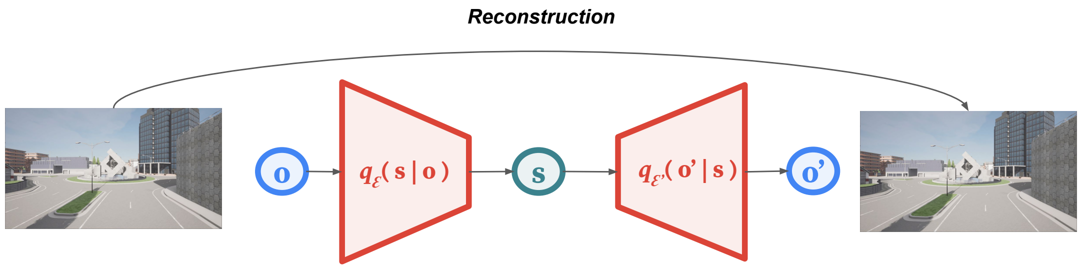

Figure 4.10: Autoencoder training for image-based task

We evaluate the performance of our method on the CARLA autonomous driving D4RL task. The task consists of an agent which has control to the throttle (gas pedal), the steering, and the brake pedal for the car. It receives 48 × 48 RGB images from the driver's perspective as observations. We use a U-net autoencoder architecture to create a 32-dimensional compressed state for this task. The horizon for LDCQ is set to $H = 30$. The results are tabulated in Table 4.6.

Table 4.6: Performance comparison on image-based CARLA task

Dataset	**BC**	**BCQ**	**CQL**	**IQL**	**LDCQ (Ours)**
carla-lane-v0	17.2	-0.1	20.9	18.6	**24.7**

4.5 Conclusion

In this work, we showed that offline RL datasets comprised of suboptimal demonstrations have expressive multi-modal latent spaces which can be captured with temporal abstraction and are 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. This method is highly suited for learning policies in safety-critical environments, as it does not rely on online training interactions. Further, the learnt model can also be used for safety checks during the execution phase. However, offline methods are significantly worse in performance when compared to online RL methods. Thus while being the best in terms of safety during training, we have to trade-off in terms of obtaining optimal policies.

Chapter 5

Conclusions

In this work, we explored innovative methods to enhance reinforcement learning in safety-critical environments. The proposed methods improved the sample efficiency, interpretability, and reliability of RL algorithms, making them suitable for training and deployment in safety critical environments. In chapter 2, we presented a novel curriculum learning method for environments where other dynamic agents are present. Our method was able to improve the sample efficiency of a nominal RL approach and also converged to a better policy. However, this method still being a reactive policy does not give interpretable trajectories. In chapter 3, we presented a hybrid method combining motion primitives with reinforcement learning for decision making that allows a high-level trajectory planning approach with reinforcement learning desired for long horizon tasks and interpretability. It improves over traditional graph-search methods and extends RL with motion primitives with a large primitives library. However, the training procedure in this case is very unstable and thus the method is very sample-inefficient. In chapter 4, we presented a novel offline reinforcement learning method for robotic control tasks which learns policies without any real-world interactions and allows learning world models which can be used for safety checks during execution. While a complete offline approach might not result in optimal policies as compared to online RL approaches, this is crucial for enabling intelligent agents for safety critical systems when collecting online data is not possible. In terms of real-time performance as well, our offline method which relies on diffusion is comparatively slower as compared to other methods. However this can be improved with using better and faster diffusion techniques not explored in this work. Hence, for safety critical environments, we are currently able to enhance RL algorithms, but there are trade-offs associated with them. A combination of these methods can be used to achieve a better algorithm.

Collectively, our research explores advancements in reinforcement learning for safety critical robotic tasks, providing a powerful frameworks to address decision-making and control problems. The proposed methods enhance sample efficiency, interpretability, and reliability, contributing to the development of more intelligent and adaptive RL agents that can be deployed in safety-critical environments.

Bibliography

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 22–31. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/achiam17a.html. 1.1

[2] 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 *NeurIPS 2022 Foundation Models for Decision Making Workshop*, . 4.1.2, 4.4.5

[3] 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*, . 4.1.1, 4.3.1, 4.4.4

[4] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019. 1.1, 2.2.2

[5] Matthias Althoff, Markus Koschi, and Stefanie Manzinger. Commonroad: Composable benchmarks for motion planning on roads. In *Proc. of the IEEE Intelligent Vehicles Symposium*, 2017. ISBN 9781509048045. doi: 10.1109/ivs.2017.7995802. 2.3.2, 3.4.3

[6] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017. 1.1

[7] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 4.4.4

[8] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1-2):41–77, 2003. 1.1, 3.1

[9] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009. 1.1, 2.1, 2.2.2

[10] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012. 3.1

[11] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, P. Abbeel, A. Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Neural Information Processing Systems*, 2021. 4.1.1, 4.4.5

[12] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018. 3.2.1

[13] Josiah Coad, Zhiqian Qiao, and John M Dolan. Safe trajectory planning using reinforcement learning for self driving. *arXiv preprint arXiv:2011.04702*, 2020. 3.2.2

[14] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. *Advances in neural information processing systems*, 5, 1992. 3.2.2

[15] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015. 3.4

[16] Jeffrey L Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1), 1993. 2.2.2

[17] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. 4.4.1, 4.4.5

[18] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018. 4.3.2

[19] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pages 2052–2062. PMLR, 2019. 1.1, 4, 4.1.1, 4.3.2, 4.4.5

[20] 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. 4.3.1, 4.4.4

[21] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014. 4.1.2

[22] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019. 3.2.1

[23] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019. 3.2.1

[24] 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. *arXiv preprint arXiv:2303.09556*, 2023. 4.3.1

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 4.4.4

[26] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018. 1.1

[27] Irina Higgins, Loïc Matthey, Arka Pal, Christopher P. Burgess, Xavier Glorot, Matthew M. Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2016. 4.2.1, 4.3.1

[28] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*. 4.3.1

[29] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020. 4.2.1, 4.3.1

[30] León Illanes, Xi Yan, Rodrigo Toro Icarte, and Sheila A McIlraith. Symbolic plans as high-level instructions for reinforcement learning. In *Proceedings of the international conference on automated planning and scheduling*, volume 30, pages 540–550, 2020. 3.2.2

[31] Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, pages 9902–9915. PMLR, 2022. 4.1.2, 4.3.3, 4.4.5

[32] Heewoo Jun and Alex Nichol. Shap-e: Generating conditional 3d implicit functions. *arXiv preprint arXiv:2305.02463*, 2023. 4.3.1

[33] Diederik P Kingma and Max Welling. Auto-encoding variational {Bayes}. In *Int. Conf. on Learning Representations*. 4.1.2, 4.2.1, 4.3.1

[34] 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*. 4.1.1

[35] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*. 1.1, 4, 4.4.5

[36] Aviral Kumar, Xue Bin Peng, and Sergey Levine. Reward-conditioned policies. *arXiv preprint arXiv:1912.13465*, 2019. 4.1.1

[37] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020. 1.1, 4, 4.1.1, 4.4.5

[38] Romain Laroche, Paul Trichelair, and Remi Tachet Des Combes. Safe policy improvement with baseline bootstrapping. In *International conference on machine learning*, pages 3652–3661. PMLR, 2019. 1.1

[39] David N Lee. A theory of visual control of braking based on information about time-to-collision. *Perception*, 5(4):437–459, 1976. 2.5.4

[40] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016. 1.1

[41] Sergey Levine, Aviral Kumar, G. Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *ArXiv*, abs/2005.01643, 2020. 4.2.2

[42] 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*, pages 11461–11471, 2022. 4.1.2

[43] Marlos C Machado, André Barreto, Doina Precup, and Michael Bowling. Temporal abstraction in reinforcement learning with the successor representation. *Journal of Machine Learning Research*, 24(80):1–69, 2023. 3.1

[44] Timothy Mann and Shie Mannor. Scaling up approximate value iteration with options: Better policies with fewer iterations. In *International conference on machine learning*, pages 127–135. PMLR, 2014. 3.1

[45] 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. 1.1

[46] Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023. 1.1

[47] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *arXiv preprint arXiv:2003.04960*, 2020. 2.1

[48] Kaleb Ben Naveed, Zhiqian Qiao, and John M. Dolan. Trajectory planning for autonomous vehicles using hierarchical reinforcement learning. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 601–606, 2021. doi: 10.1109/ITSC48978.2021.9564634. 3.2.2

[49] Karl Pertsch, Youngwoon Lee, and Joseph Lim. Accelerating reinforcement learning with learned skill priors. In Jens Kober, Fabio Ramos, and Claire Tomlin, editors, *Proceedings of the 2020 Conference on Robot Learning*, volume 155 of *Proceedings of Machine Learning Research*, pages 188–204. PMLR, 16–18 Nov 2021. URL `https://proceedings.mlr.press/v155/pertsch21a.html`. 4.1.1, 4.3.1

[50] Zhiqian Qiao, Katharina Muelling, John M. Dolan, Praveen Palanisamy, and Priyantha Mudalige. Automatically generated curriculum based reinforcement learning for autonomous vehicles in urban environment. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1233–1238, 2018. doi: 10.1109/IVS.2018.8500603. 2.2.2

[51] Duy Quang Tran and Sang-Hoon Bae. Proximal policy optimization through a deep reinforcement learning framework for multiple autonomous vehicles at a non-signalized intersection. *Applied Sciences*, 10(16):5722, 2020. 2.5

[52] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022. 4, 4.1.1, 4.1.2, 4.3.1

[53] 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*, pages 10684–10695, 2022. 4, 4.2.1

[54] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. volume 9351, pages 234–241, 10 2015. ISBN 978-3-319-24573-7. doi: 10.1007/978-3-319-24574-4_28. 4.1.2, 4.4.4

[55] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan,

Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, 35:36479–36494, 2022. 4, 4.1.1, 4.1.2

[56] Chitwan Saharia, Jonathan Ho, William Chan, Tim Salimans, David J Fleet, and Mohammad Norouzi. Image super-resolution via iterative refinement. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022. 4.1.2

[57] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015. 4.3.2

[58] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020. 1.1

[59] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 3.2.2

[60] Moritz Klischat Sebastian Maierhofer and Matthias Althoff. Commonroad scenario designer: An open-source toolbox for map conversion and scenario creation for autonomous vehicles. In *Proc. of the IEEE Int. Conf. on Intelligent Transportation Systems*, pages 3176–3182, 2021. doi: https://doi.org/10.1109/itsc48978.2021.9564885. 2.3.1

[61] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016. 1.1, 3.1

[62] 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`. 4.1.2, 4.2.1

[63] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019. 4.1.2, 4.2.1

[64] Yunlong Song, HaoChih Lin, Elia Kaufmann, Peter Dürr, and Davide Scaramuzza. Autonomous overtaking in gran turismo sport using curriculum reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9403–9409. IEEE, 2021. 2.2.2

[65] 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. 4.1.1

[66] Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier, 1990. 3.2.1

[67] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 4.3.2

[68] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 4.1.1

[69] Sushant Veer and Anirudha Majumdar. Probably approximately correct vision-based planning using motion primitives. *arXiv preprint arXiv:2002.12852*, 2020. 3.2.2

[70] Nino Vieillard, Marcin Andrychowicz, Anton Raichuk, Olivier Pietquin, and Matthieu Geist. Implicitly regularized rl with implicit q-values. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 1380–1402. PMLR, 28–30 Mar 2022. URL `https://proceedings.mlr.press/v151/vieillard22a.html`. 4.1.1

[71] Xiao Wang, Hanna Krasowski, and Matthias Althoff. Commonroad-rl: A configurable reinforcement learning environment for motion planning of autonomous vehicles. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2021. doi: 10.1109/ITSC48978.2021.9564898. 2.1, 3.1

[72] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020. 2.2.1