Safe and Trustworthy Decision Making through Reinforcement Learning

By

Jinning Li

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering Science - Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Masayoshi Tomizuka, Chair
Professor Roberto Horowitz
Professor Francesco Borrelli

Spring 2024

Safe and Trustworthy Decision Making through Reinforcement Learning

Abstract

Safe and Trustworthy Decision Making through Reinforcement Learning

by

Jinning Li

Doctor of Philosophy in Engineering Science - Mechanical Engineering

University of California, Berkeley

Professor Masayoshi Tomizuka, Chair

The advent of advanced computational technologies and artificial intelligence has ushered in a new era of complex systems and applications, notably in the realms of autonomous vehicles (AVs) and robotics. These systems are increasingly required to make decisions autonomously in dynamic and uncertain environments. Reinforcement Learning (RL) has emerged as a pivotal technique in this context, offering a framework for learning optimal decision-making strategies through interactions with the environment. However, ensuring safety and trustworthiness in these decisions remains a critical challenge, especially in safety-critical applications such as autonomous driving.

This dissertation addresses the aforementioned challenge by proposing innovative RL-based approaches, and is structured into three distinct but interconnected parts, each focusing on a unique aspect of RL in the context of safe and trustworthy decision-making. The thread of this dissertation is based on the exploration and advancement of RL techniques to ensure safety and reliability in autonomous decision-making systems, particularly in complex, dynamic environments.

We first establish the foundational aspects of RL in decision-making, particularly in uncertain and dynamic environments. The focus here is on enhancing RL to deal with real-world complexities, such as interacting with unpredictable agents, e.g., human drivers in AV scenarios, and handling distributional shifts in offline RL settings. This sets the stage for understanding and improving the decision-making capabilities of autonomous systems under uncertainty.

Building on the first part, we then explore the integration of hierarchical planning with RL. The emphasis is on creating frameworks that combine different levels of decision-making, balancing immediate, low-level safety concerns with high-level strategic objectives. The approach aims to address the limitations of traditional RL in complex, multi-agent environments and long-duration tasks, demonstrating improved adaptability and efficiency in

real-time decision-making.

The final part represents a forward-looking approach to RL, focusing on the integration of offline and online learning methodologies. This part addresses the challenge of training RL agents in a manner that is both safe and effective, particularly in contexts where exploration can be costly or dangerous. By combining the strengths of large-scale offline data (expert demonstrations) with online learning, we present a novel framework for enhancing the safety and performance of RL agents in practical, real-world applications.

To My Family

Contents

List of Figures

List of Tables

Acknowledgments

The past five years have been a joyful journey, filled with learning, exploration, and personal growth. As I reflect on this journey, I am filled with gratitude for the numerous individuals who have supported, guided, and inspired me along the way. Their contributions have been instrumental in shaping not only this dissertation but also my growth as a scholar and individual. It is with a deep sense of appreciation that I acknowledge their roles in this significant phase of my life.

I would like to express my deepest gratitude to my PhD advisor, Professor Masayoshi Tomizuka, for his invaluable guidance, unwavering support, and insightful critiques throughout the course of this research. His expertise and mentorship have been pivotal in shaping both this dissertation and my development as a researcher.

I am also immensely thankful to my dissertation committee members, Professor Masayoshi Tomizuka, Professor Roberto Horowitz, and Professor Francesco Borrelli for their constructive feedback, encouragement, and rigorous examination of my work. Their perspectives and suggestions have greatly contributed to the depth and quality of this research.

I would like to extend my appreciation to Dr. Chen Tang, Dr. Wei Zhan, Dr. Liting Sun, Dr. Jiachen Li, Dr. Banghua Zhu, Dr. Jiantao Jiao, Dr. Jianyu Chen, Dr. Hengbo Ma, Yuxin Chen, Chenran Li, Ran Tian, Xinyi Liu for their collaboration, stimulating discussions, and the sharing of ideas which have enriched my research experience. Working alongside them has been both an honor and a privilege.

Special thanks go to all my colleagues at Mechanical Systems Control Lab for creating a vibrant and collaborative research environment. The camaraderie and support within our team have been a source of motivation and inspiration.

I am indebted to DENSO International America, and Momenta for providing the financial support that made this research possible. Their belief in the potential of this project has been greatly encouraging.

On a personal note, I wish to express my heartfelt gratitude to my parents, W. Li, and J. Li. Their love, sacrifice, and unwavering belief in my abilities have been my constant source of strength and resilience. Dr. Ruochen Yang, your support and understanding have been my anchor during the most challenging times of this journey.

Finally, I would like to thank my friends, both within and outside the academic community, for their invaluable support, encouragement, and for always being there to offer a listening ear or a word of advice.

This dissertation stands as a testament to the collective efforts and encouragement of all those mentioned above, and many others whose contributions have been just as significant. Thank you.

Chapter 1

Introduction

In an era where technology profoundly influences every aspect of life, the advent of intelligent systems capable of autonomous decision-making heralds a transformative shift in various domains, from autonomous vehicles to robotics. Central to the advancement of these systems is the development of robust and reliable decision-making algorithms that not only optimize performance but also ensure safety and trustworthiness. This dissertation explores the frontier of reinforcement learning (RL), a powerful paradigm for learning decision-making strategies through interaction with the environment, to address these critical challenges.

1.1 Challenges of Safe and Trustworthy Reinforcement Learning

The burgeoning field of autonomous systems, epitomized by autonomous vehicles (AVs), has brought to the fore the critical challenge of applying Reinforcement Learning (RL) in real-world scenarios, where safety and trustworthiness are paramount. In the dynamic and unpredictable world of road traffic, AVs encounter a myriad of situations fraught with uncertainties - unpredictable human behavior, rapidly changing conditions, and a diverse array of interactive scenarios. These complexities necessitate a decision-making paradigm that is robust, adaptable, and above all, safe.

Traditional RL methodologies, while effective in controlled environments, are often ill-equipped for real-world applications. There are still challenges unsolved for the application of these RL methods. In summary, safe and trustworthy reinforcement learning in real-world scenarios confront the following challenges:

The first major challenge, as discussed in the first project, involves behavior planning for AVs amidst interactive uncertainties in mixed-traffic environments. AVs must share roads with human-driven vehicles, each exhibiting a range of behaviors from cooperative to aggressive, attentive to inattentive. This diversity in behavior patterns creates a scenario of high uncertainty, necessitating AVs to not only predict but also adapt to various human responses. For instance, an AV's decision to change lanes requires anticipation of whether nearby drivers will yield or not, which significantly varies depending on their level of cooperativeness and attention.

The problem of distributional shift is a major challenge in offline RL. In real-world applications like autonomous driving, actively collecting data can be costly or dangerous. Offline RL, where policies are learned from static datasets, becomes crucial but introduces the issue of distributional shift — the difference in state and action distributions between the training dataset and real-world deployment. This shift can lead to policies that perform well in training but fail dramatically in real-world scenarios, especially in safety-critical situations.

It is challenging to balance safety with efficiency in various scenarios. Traditional single-layer optimization or sampling-based motion planners struggle to generate safe trajectories in real-time, especially in dense traffic with multiple interactive vehicles. Conversely, end-to-end learning methods often cannot assure the outcomes' safety. The hierarchical approach, integrating low-level safe controllers with a high-level RL algorithm, provides a solution. This strategy ensures safety through low-level controllers while the high-level RL algorithm adapts the behavior planning, addressing the complexity of real-world traffic scenarios where a one-size-fits-all approach is insufficient.

Another key challenge arises in temporally extended tasks, common in safety-critical applications like autonomous driving, where exploration can be risky and costly. These tasks require a level of foresight and planning beyond the capabilities of standard RL approaches. The safe and efficient navigation of AVs in these complex, time-extended scenarios is a test of the adaptability and robustness of RL methodologies.

Integrating offline and online training in safe RL to take advantage of both training scheme is promising, but challenges also arise from the integration. Traditional safe RL algorithms tend to be overly conservative when learning from scratch, limiting exploration and hindering performance. In real-world tasks like autonomous driving, where large-scale expert demonstration data are available, utilizing this data effectively becomes crucial.

In summary, the challenges of implementing safe and trustworthy RL in real-world scenarios is multifaceted. From navigating interactive uncertainties and addressing distributional shifts in offline settings to balancing safety with efficiency in hierarchical planning and integrating offline and online training methods, these challenges reflect the complexity and dynamic nature of real-world applications of RL.

1.2 Safe Reinforcement Learning Methods for Planning

This dissertation addresses these challenges through a multifaceted approach that integrates innovative RL methodologies and planning structures, specifically tailored for real-world decision-making. The strategy is rooted in a deep understanding of the complexities and requirements of real-world scenarios, particularly those involving AVs.

For behavior planning in AVs, this dissertation explores the use of Partially Observable

Markov Decision Processes (POMDPs) to effectively navigate the uncertainties inherent in road traffic scenarios. POMDPs provide a framework for making informed decisions even in the presence of incomplete information, which is crucial for ensuring the safety and efficiency of AVs amidst unpredictable human behavior.

In tackling the issue of distributional shift in offline RL, the dissertation introduces the Pessimistic Offline Reinforcement Learning (PessORL) algorithm. This innovative approach strategically handles out-of-distribution states by penalizing high-value predictions absent in the training dataset, thereby enhancing the performance and reliability of RL agents in real-world deployment.

The dissertation also explores hierarchical planning, proposing a novel framework that synergizes low-level safe controllers with high-level RL algorithms. This approach effectively balances immediate safety concerns with strategic decision-making, ensuring adaptability and efficiency in complex traffic conditions.

Additionally, the dissertation takes a forward-looking approach by exploring the integration of offline and online RL training methodologies, particularly through the Guided Online Distillation (GOLD) framework. This innovative method leverages the strengths of large-scale offline data and online learning, optimizing the reward-cost trade-off in safety-critical scenarios and enhancing the decision-making capabilities of RL agents.

1.3 Dissertation Outline

The dissertation is structured into three parts, each focusing on a distinct aspect of RL in the context of safe and trustworthy decision-making:

1. **Part One: Reinforcement Learning for Decision Making.** Chapter 2 is focused on behavior planning for AVs in mixed-traffic environments using POMDPs, addressing the need for AVs to adapt to the unpredictable behaviors of other road users. Chapter 3 introduces the PessORL algorithm, an innovative solution to the distributional shift problem in offline RL, enhancing agent performance in real-world deployment.

2. **Part Two: Hierarchical Planning Based on Reinforcement Learning.** Chapter 4 proposes a hierarchical behavior planning framework, integrating low-level safe controllers with high-level RL to navigate complex traffic scenarios efficiently and safely. Chapter 5 extends hierarchical planning to offline RL for temporally extended tasks, showcasing the framework's effectiveness in long-horizon driving and robot navigation tasks.

3. **Part Three: Looking Forward - Combining Offline and Online Training.** Chapter 6 explores the innovative GOLD framework, combining offline and online RL training methods to improve decision-making in safety-critical scenarios. This approach addresses the limitations of conservative exploration in safe RL and leverages expert demonstrations for enhanced policy learning.

Each part and its respective projects contribute to the overarching objective of developing RL methodologies that enable safe, efficient, and reliable autonomous decision-making in the real world. This dissertation not only tackles key theoretical and practical challenges in RL but also lays a foundation for future research in the field of autonomous systems and robotics.

Part I

Reinforcement Learning for Decision Making

Chapter 2

Interaction-Aware Behavior Planning for Autonomous Vehicles Validated with Real Traffic Data

Autonomous vehicles (AVs) need to interact with other traffic participants who can be either cooperative or aggressive, attentive or inattentive. Such different characteristics can lead to quite different interactive behaviors. Hence, to achieve safe and efficient autonomous driving, AVs need to be aware of such uncertainties when they plan their own behaviors. In this chapter, we formulate such a behavior planning problem as a partially observable Markov Decision Process (POMDP) where the cooperativeness of other traffic participants is treated as an unobservable state. Under different cooperativeness levels, we learn the human behavior models from real traffic data via the principle of maximum likelihood. Based on that, the POMDP problem is solved by Monte-Carlo Tree Search. We verify the proposed algorithm in both simulations and real traffic data on a lane change scenario, and the results show that the proposed algorithm can finish the lane changes with high success rates.

2.1 Introduction

Although rapid progress has been made in autonomous driving in the past decade, there are many challenges yet to be solved. One of such challenges lies in the behavior planning of autonomous vehicles (AVs) in the presence of uncertainties introduced during interactions with other traffic road participants. To safely and efficiently share roads with other road users, AVs need to plan their own behaviors while considering the interactive behaviors of others via behavior prediction [12, 153, 84]. For instance, an autonomous vehicle should predict and reason about what other vehicles might respond if it accelerates to finish a take-over. A cooperative and attentive driver may yield, while an aggressive or inattentive driver may not. Depending on their different cooperativeness levels, the autonomous vehicles should behave differently to assure safety and efficiency.

Many research efforts have been attracted to the behavior planning problem under interactive uncertainties. These approaches can be categorized into three groups. The first group is direct policy learning via reinforcement learning (RL) [99, 145, 86, 83]. For example, [99] utilized passive actor-critic (pAC) in RL to generate interactive policies, while [145] used Deep Deterministic Policy Gradient for policy learning. The second group is to formulate the interaction-aware behavior planning as a partially observable Markov Decision Process (POMDP) [84]. Namely, the uncertainties will first be estimated via observations for better decision-making. Many different hidden variables have been proposed to represent different uncertainties, including sensor noises [140, 64], occlusions [132, 2, 10], human intentions [58, 80, 125, 7], and cooperativeness of humans [37], to name a few. To reduce the computation load of POMDP, the authors of [30] proposed a multi-policy Decision Making (MPDM) method to simplify the POMDP problem.

However, the first two groups do not explicitly consider the mutual influence between the behaviors of AVs and other road users. Namely, in terms of the POMDP formulation, the estimation of the hidden variables (uncertainties) is assumed to depend only on the historical states of interactive agents, i.e., not influenced by the potential behaviors of AVs. Such an assumption, however, fails to capture the fact that the intentions or the cooperativeness of other agents can be influenced by the AVs behaviors since other agents are (approximately) rational agents instead of non-intelligent obstacles. Therefore, the third group formulates the interaction-aware behavior planning problem by explicitly leverage such influence. For example, [120] formulated the problem as a nested optimization problem assuming that the other agents always respond to the AVs with their best responses. [33] proposed cooperative behavior planning approaches. In [107], Peng et al. proposed to use Bayesian persuasion model to model the "negotiation" process of interactions. In [57], the authors represented the human behaviors via heuristic low-level driver models (intelligent driver model) that is influenced by the potential actions from the AVs.

All the approaches mentioned above have been verified for effectiveness in one or multiple driving scenarios via simulations. However, most of them have not been verified with real traffic data. In this chapter, we aim to propose an interaction-aware behavior planning algorithm with validation on real traffic data. Focusing on a lane-changing scenario, we also formulate the problem as addressed in the third category. More specifically, similar to [57], we represent the continuous-time trajectories of the other agents via low-level driver models, and estimate the cooperativeness of humans considering the influence from future actions of the AVs. To generate more human-like interactive behaviors, we learn the optimal distribution of the low-level driver models from data to capture the diversity of different humans. Based on the learned models and the estimated cooperativeness, we solve the behavior planning problem via Monta-Carlo Tree Search (MCTS). Finally, the proposed approach is verified in both simulation and real traffic data. The results show that the proposed planner can generate actions for the AVs to successfully execute the lane-changing task. Statistical results on the real traffic data also show that the proposed planner with the learned model can generate more human-like behaviors compared to those with heuristic driver models.

2.2 Human Behavior Modelling

Vehicle Models

Throughout the report, we focus on the lane-changing scenarios. Three vehicles will be considered: one AV (the ego agent N_0) and two surrounding agents on the target lane: one following vehicle N_1 and one leading vehicle N_2 driving. The goal of the ego vehicle is to merge in between the two human vehicles N_1 and N_2 on the target lane. Therefore, in most cases, the leading vehicle N_1 is not affected by the maneuvers of the ego vehicle. Only the following vehicle N_2 has to decide whether or not to yield to the ego vehicle, and hence the ego vehicle needs to predict the behavior of the following vehicle to make reasonable decisions on its behalf.

Driver models estimate the behavior of the human cars on the roadway based on the current state of the car and the car immediately in front of it. The driver models predict the acceleration of human cars in the target lane based on the state of each of the vehicles. We denote the car's acceleration, $\hat{a}$, as

$$\hat{a} = f(x; \theta), \tag{2.1}$$

where the function $f(x;\theta)$ denotes the driver model function of the states x and the model parameters θ. Researchers have done plenty of prior works on human vehicles model and prediction with complex algorithm design [154, 55, 91]. However, since this work is focused on the design of the planning module, we adopt and compare Intelligent Driver Model, Velocity Driver Model, and Smart Driver Model for simplicity and efficiency.

Intelligent Driver Model

The Intelligent Driver Model (IDM) [138] is given by Eqns. 2.2 and 2.3. The model describes the acceleration a of the back car, as a function of the car's velocity v_{back}, the reference velocity v_0, the difference between the car velocity and the velocity of the car in front $\Delta v = v_{back} - v_{front}$, and the following distance $\varphi = s_{front} + N_{length,front} - s_{back}$. Here, s_{front} is the position of the front car, s_{back} denotes the position of the back car, and $N_{length,front}$ denotes the length of the front car.

$$a = a_{max}\left[1 - \left(\frac{v_{back}}{v_0}\right)^{\delta} - \left(\frac{\phi\left(v_{back}, \Delta v\right)}{\varphi}\right)^2\right] \tag{2.2}$$

$$\phi\left(v_{back}, \Delta v\right) = d_0 + v_{back}T + \frac{v_{back}\Delta v}{2\sqrt{ab}} \tag{2.3}$$

The physical interpretation of the parameters are the minimum following time, T, the minimum following distance, d_0, the maximum acceleration, a, and the braking deceleration, b.

The IDM is represented using the state vector x and parameter vector θ per Eq. 2.1, where x and θ are defined by

$$x = \begin{bmatrix} v_{back} & \Delta v & \varphi \end{bmatrix}^T \tag{2.4}$$

$$\theta = \begin{bmatrix} T & a_{max} & v_0 & \delta & d_0 & b \end{bmatrix}^T. \tag{2.5}$$

Velocity Difference Model

The Velocity Difference Model [63] is given in Eq. 2.6 and 2.7.

$$a = \kappa[V(s) - v_{back} + \lambda \Delta v] \tag{2.6}$$

where κ is a sensitivity constant and V is the optimal velocity that the drivers prefer. Researchers adopted an optimal velocity function as

$$V(s) = V_1 + V_2 \tanh[C_1 \varphi - C_2] \tag{2.7}$$

where $\varphi = s_{front} + N_{length,front} - s_{back}$, $\Delta v = v_{back} - v_{front}$. Similar to the setting aforementioned, we use the state vector x and parameter vector θ as

$$x = \begin{bmatrix} v_{back} & \Delta v & \varphi \end{bmatrix}^T \tag{2.8}$$

$$\theta = \begin{bmatrix} V_1 & V_2 & C_1 & C_2 & \lambda & \kappa \end{bmatrix}^T. \tag{2.9}$$

Smart Driver Model

The Smart Driver Model (SDM) [90] is given by Eq. 2.10 and Eq. 2.11. The physical meaning of the parameters and variables are the same as those in IDM.

$$a = a_{max} \left[1 - \left(\frac{v_{back}}{v_0} \right)^{\delta} \right] - a_{dec} \tag{2.10}$$

$$a_{dec} = \frac{a_{max} \left[1 - \left(\frac{v_{back}}{v_0} \right)^{\delta} \right] + \frac{v_{back}^2 - v_{front}^2}{2\varphi}}{\exp\left(\frac{\Delta x}{d_0 + v_{back} \times T} - 1 \right)} \tag{2.11}$$

Eq. 2.11 is used to describe the deceleration term. This model is similar to the IDM but it can improve the stability compared to IDM under homogenous traffic condition. In this chapter, the SDM considers only the single-lane scenario and ignores the lane changing problem.

Following Eq. 2.1, we represent the SDM using the state vector x and parameter vector θ as

$$x = \begin{bmatrix} v_{back} & \Delta v & \varphi \end{bmatrix}^T \tag{2.12}$$

and

$$\theta = \begin{bmatrix} T & a_{max} & v_0 & \delta & d_0 \end{bmatrix}^T. \tag{2.13}$$

Parameters Calibration

Real Traffic Dataset

To increase applicability of planning algorithms for autonomous driving, a crucial step is to design and evaluate the algorithm on realistic traffic dataset. The INTERACTION [156] is a publicly available driving dataset suitable for testing our planning method. It delivers pertinent properties for designing autonomous driving planning algorithms including:

- Diversity of interactive driving scenarios;
- International driving cultures;
- Complexity of the scenarios and behavior;
- Criticality of the situations;
- Map information;
- Completeness of interaction entities.

We propose to adopt the INTERACTION dataset as the realistic environment to evaluate our designed planning method for merging scenarios. We use a data set collected by drones from a highway ramp merging scenario. It contains the position, the velocity and the length of each car in the scene. We pick up the data of similar scenarios, with an ego car merging to the neighbor lane.

Calibration Implementation

We assume the actual acceleration of the back human car on the target lane is given by the predicted acceleration and additive Gaussian noise (Eq. 2.14).

$$a = f(x;\theta) + \epsilon \qquad \epsilon \sim \mathcal{N}\left(0, \sigma_\epsilon^2\right) \tag{2.14}$$

Equivalently to Eq. 2.14, the behavior is described by Eq. 2.15 as a Gaussian probability density function of acceleration a conditional on the states x and parameters θ.

$$p\left(a|x;\theta,\sigma_\epsilon\right) = \frac{1}{\sqrt{2\pi}\sigma_\epsilon} e^{-\frac{(a-\hat{a}(x,\theta))^2}{2\sigma_\epsilon^2}} \tag{2.15}$$

A sequences of recorded m sets of the state and acceleration, $\{(x^{(1)}, a^{(1)}), (x^{(2)}, a^{(2)}), ..., (x^{(m)}, a^{(m)})\}$, are recorded over time for a back and a front human car from the data. We employ the following Maximum a Posteriori estimation method to estimate the model parameters, θ, using the recorded values of x and a. The method is set up to minimize a cost function $J(\theta; x, a)$ to find the optimal parameters θ (Eq. 2.16).

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\ J(\theta; x, a) \tag{2.16}$$

We use a data set collected by drones from a highway ramp merging scenario [155]. It contains the position, the velocity and the length of each car in the scene. We process the data set to pick up the data of similar scenarios (an ego car merging to the neighbor lane).

The data is divided into two sets, successful trials, where the merging car is successfully able to merge between the front and back car, and unsuccessful trials, merging car must fall behind the back car in order to merge into the lane. In the unsuccessful case, the back car is directly behind the front car, so its acceleration is predicted by the vehicle models using front car as the front car's state. Conversely, in the successful case, the merging car drives between the front and back car, so the acceleration of the back car is no longer calculated from the front car. Instead, the vehicle models use the state of the merging car to predict the acceleration of the back car.

The sets contain $p = 75$ trials for the unsuccessful case, and $p = 115$ trials for the successful case. Each trial contains m time steps of data, which varies depending on the trial (around 5 seconds with 100 recordings per second).

The vehicle model parameters are calibrated separately for each trial, since each trial represents a different car. Here the goal is to find a distribution of the model parameters fitting the set of parameters $\{\theta_1, \theta_2, \theta_3, \ldots, \theta_p\}$ which we obtained from maximum likelihood estimation, so that the variation in driver behavior can be captured. For each set (successful/unsuccessful), the parameter estimation is found using Maximum Likelihood Estimation method. Outliers are excluded on the basis of the returned value of the cost function, $J(\theta^*; x, a)$; i.e. if $J(\theta^*; x, a)$ of the trial is above a limit α it is excluded from analysis. The limit α is set according to $\alpha = Q_3 + 1.5IQR$, where Q_3 represents the third quartile of the cost function evaluation for the trials and IQR represents the respective interquartile range.

- Least Squared Error
  The problem is formulated as a least-squares problem with the cost function

$$J(\theta; x, a) = \frac{1}{m} \sum_{i=1}^{m} \left(a^{(i)} - \hat{a}(x^{(i)}, \theta)\right)^2, \tag{2.17}$$

  where $\hat{a}(x^{(i)}, \theta)$ is calculated from Eq. 2.1. The goal is to minimize the square of the prediction error between the actual acceleration, a, and the acceleration predicted by the model, $\hat{a}$.

- Maximum Likelihood

  The maximum likelihood method attempts to maximize the log-likelihood of the parameters θ given the measured states x. For a conditional Gaussian, we find the log-likelihood by taking the log of the likelihood $l(\theta; x, a) = p(a|x; \theta)$ in Eq. 2.15. The log-likelihood is given by Eq. 2.18, where the C represents a constant term that is independent of θ and x and therefore does not affect the optimization problem.

$$\log l(\theta; x, a) = C - \frac{1}{2\sigma_\epsilon^2} \sum_{i=1}^{m} \left(a^{(i)} - \hat{a}(x^{(i)}, \theta)\right)^2 \tag{2.18}$$

- Maximum a Posteriori

Maximum a posteriori incorporates prior knowledge about the distribution of the parameters into the maximum likelihood formulation. A Gaussian distribution of the parameters is assumed a priori:

$$P(\theta; \mu_0, \Sigma_0) = \mathcal{N}(\mu_0, \Sigma_0). \tag{2.19}$$

Here μ_0 and Σ_0 represent the a priori mean and covariance matrix of the parameters θ, and were set based on previous literature and the physical interpretation/limits of each parameter. The parameters were assumed to be independent random variable. The assumed means and standard deviations are given in Table 2.1.

The loglikelihood of this a priori term is given by

$$\log P(\theta; \mu_0, \Sigma_0) = -\frac{1}{2} (\theta - \mu_0)^\top \Sigma_0^{-1} (\theta - \mu_0). \tag{2.20}$$

Model	**Parameters**	**Mean μ_0**	**Covariance Σ_0**
θ_{IDM}	$\begin{bmatrix} T \\ a_{max} \\ v_0 \\ \delta \\ d_0 \\ b \end{bmatrix}$	$\begin{bmatrix} 0.85 \\ 1.5 \\ 20 \\ 4 \\ 2.9 \\ 2.5 \end{bmatrix}$	$\mathrm{diag}\left(\begin{bmatrix} 0.35 \\ 0.25 \\ 7 \\ 0.5 \\ 1.3125 \\ 0.5 \end{bmatrix}\right)$
θ_{SDM}	$\begin{bmatrix} T \\ a_{max} \\ v_0 \\ \delta \\ d_0 \end{bmatrix}$	$\begin{bmatrix} 0.85 \\ 1.5 \\ 20 \\ 4 \\ 2.9 \end{bmatrix}$	$\mathrm{diag}\left(\begin{bmatrix} 0.35 \\ 0.25 \\ 7 \\ 0.5 \\ 1.3125 \end{bmatrix}\right)$
θ_{VDM}	$\begin{bmatrix} V_1 \\ V_2 \\ C_1 \\ C_2 \\ \lambda \\ \kappa \end{bmatrix}$	$\begin{bmatrix} 6.75 \\ 7.91 \\ 0.13 \\ 1.57 \\ 1.00 \\ 0.41 \end{bmatrix}$	$\mathrm{diag}\left(\begin{bmatrix} 2.75 \\ 2.41 \\ 0.23 \\ 3.43 \\ 0.50 \\ 0.30 \end{bmatrix}\right)$

Table 2.1: Assumed a priori parameter distributions for MAP formulation

This term is incorporated into the cost function in Eq. 2.21, where $\log l(\theta; x)$ is found from Eq. 2.18.

$$J = -\log P(\theta; \mu_0, \Sigma_0) - \log l(\theta; x, a) \tag{2.21}$$

2.3 Interaction-Aware Behavior Planning

As described in Sec. 2.2, we focus on the lane-changing scenarios. Three vehicles will be considered: one AV (the ego agent N_0) and two surrounding agents on the target lane: one following vehicle N_1 and one leading vehicle N_2 driving. The goal of the ego vehicle is to merge in between the two human vehicles N_1 and N_2 on the target lane.

We discretize the behavior/action space of the ego agent. To model the trajectories of the surrounding agents, we utilize hierarchical representations, similar to [57]. Namely, we represent their continuous trajectories by a high-level discrete model decision and a collection of stochastic low-level driver models. Details regarding the formulation are given below.

Partially Observable Markov Decision Process

A Partially Observable Markov Decision Process (POMDP) can be defined as a tuple $< \mathcal{X}, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{Z}, \mathcal{R}, b_0, \gamma >$, where $x \in \mathcal{X}$ is the state of the agents, $o \in \mathcal{O}$ is the observation perceived by the agents, and $a \in \mathcal{A}$ is the action taken by the agents. The transition model $\mathcal{T}(x, a, x')$ is the probability of the agents ending in the state x' when in the state x and taking the action a. The observation model $\mathcal{Z}(x', a, o)$ is the probability of the agents obtaining the observation o after it executes the action a and ends in the state x'. $\mathcal{R}(x, a)$ is the reward of taking the action a in the state $x.b_0$ is the initial belief state of the agents, and γ is the discount factor. The discount factor makes it possible to favor immediate rewards over rewards in the future.

Unlike the Markov Decision Process (MDP) where we want to find a mapping $\pi : x \mapsto a$, the policy in a POMDP is a mapping $\pi : b \mapsto a$ where it gives an action a for a given belief state b. The overall objective is to maximize the expected cumulative discounted reward and find the corresponding optimal policy

$$\pi^* = \arg\max_{\pi} E\left[\sum_{t=0}^{\infty} \gamma^t R(x_t, \pi(b_t))\right] \tag{2.22}$$

State Space

We use Frenét Frame to express the position, the velocity, and the acceleration of the three vehicles. For $k = \{0, 1, 2\}$, the position and the velocity of the vehicle N_k are denoted by s_k and v_k, respectively. d_k denotes the distance deviation from N_k to the center line of its lane. l_k is the lane number of the vehicle N_k. The state of the ego car is denoted by $x_0 = (s_0, d_0, v_0, l_0)^\top$. The state of the other car is defined as $x_k = (s_k, v_k, l_k, m_k)^\top, k \in \{1, 2\}$, where m_k is a hidden state indicating the conservatism of that car. The hidden state m_k cannot be measured directly, but can be estimated based on observations. The uncertainty in the proposed POMDP comes in here, as we add the hidden state m_k into the state space. If m_k is 1, then that car is willing to yield to the ego car, whereas if m_k is 0, then it will not drive cooperatively, not yielding to the ego car. This hidden state m_k induces a belief state space in POMDP. It affects the structure of the driver model in our algorithm, which predicts the human cars' behavior. Our algorithm then plans the next action according to its prediction.

Action Space

Similar to [57], the action space of our framework also consists of two dimensions: the longitudinal acceleration, and the lateral velocity of the ego vehicle. We simplify the actions to discrete actions, aiming to ease the curse of dimension in POMDP. The planned action would be sent to lower level planners to be smoothed so as to ensure the comfort. For the longitudinal acceleration, $A_{long} = \{-1.5m/s^2, 0m/s^2, 1m/s^2\}$. For the lateral velocity, $A_{lat} = \{-0.5m/s, 0m/s, 0.5m/s\}$. Here, we assume the ego car can achieve the lateral velocity instantly (relative to one time step), thus it makes sense to use velocity as actions.

Transition model

The transition model is defined by a set of discrete difference equations as follows:

$$s'_k = s_k + v\Delta t + \frac{1}{2}a_k\Delta t^2 \qquad k \in \{0,1,2\} \tag{2.23}$$

$$v'_k = v_k + \Delta t a_k \qquad k \in \{0,1,2\} \tag{2.24}$$

$$l'_k = l_k \pm 1 \qquad k \in \{0\} \tag{2.25}$$

$$l'_k = l_k \qquad k \in \{1,2\} \tag{2.26}$$

$$d'_k = d_k v_{k,lat} + (l'_k - l_k)w_{lane} \qquad k \in \{0\} \tag{2.27}$$

$$m'_k = m_k \qquad k \in \{1,2\} \tag{2.28}$$

where w_{lane} is the width of the current lane of the cars, and Δt is the sampling time. The left hand sides of Eq. 4-9 are values at the next time step. In our problem setting, we assume that the other two human cars would not change their lanes. We want the ego car to change to the lane of the other two cars. Therefore, the action of the ego car a_0 is obtained by solving the POMDP, and the actions of the other two human cars are predicted by driver models when solving the POMDP.

Observation Space

We assume that the ego car has access to all data except for the hidden state m_k from the sensors. Therefore, the observation of the ego car is defined as $o_0 = (s'_0, d'_0, v'_0, l'_0)^\top$. The observation of the other cars is defined as $o_k = (s'_k, v'_k, l'_k)^\top, k \in \{1, 2\}$. Here, the hidden state m_k is not presented in the observation space. It needs to be estimated by a logistic regression classifier during the planning process. The classifier is presented in the following section.

Reward Function

We adopt the reward function of our framework from the definition in [57]. The reward is a sum of several different rewards, shown as Eq. 2.29. The parameters were tuned and worked well in practice.

$$r(x, a, x') = r_{vel} + r_{act} + r_{end\_lane} + r_{wrong\_lane} + r_{center} + r_{collision} \tag{2.29}$$

r_{vel}, which is defined in Eq. 2.30 and Eq. 2.31, denotes the reward for the deviation from the reference velocity. We choose a reference velocity based on traffic rules and vehicle dynamics, and give it to the ego car to follow.

$$r_{vel} = -100(v_{ref} - v_0)^2, \qquad \text{if} \;\; v_0 > v_{ref} \tag{2.30}$$

$$r_{vel} = -100(v_{ref} - v0), \qquad \text{if} \;\; v_0 < v_{ref} \tag{2.31}$$

r_{wrong_lane} is the wrong lane reward. If the ego car is not driving on the target lane, then $r_{wrong\_lane} = -10000$. The end of lane reward r_{end_lane} encourages the ego car to change its lane before it reaches the end of the road. r_{end_lane} is negative and it decreases from 0 to -1000 linearly over the last 50m of the road.

r_{center} is the reward for the deviation from the center line of the target lane. It encourages the ego car to merge into the target lane, instead of driving on its original lane. Thus, it is defined to be $r_{center} = -200d_0^2$.

r_{act} encourages the ego car to take an action that can result in better comfort. It penalizes big value of longitudinal acceleration and latitudinal velocity with $r_{act} = -100(a_{0,long}^2 + 2|v_{0,lat}|)$.

A collision in autonomous vehicles planning is intolerant, so the reward for collision should be extremely negative, with $r_{collision} = -1000000$.

Yielding Classifier

When solving the POMDP, we need to predict the future actions of the other two human cars. We use popular driver models to make this prediction. Predicting actions of some car often involves choosing a suitable driver model for it, so we need a mechanism to make this choice. We use a yielding classifier to estimate how likely the human car is going to yield to the ego car. Then the suitable driver model for the human car is selected based on the probability of yielding. The yielding classifier is a logistic regression classifier trained by real-world data set. The probability of the human car yielding to the ego car is expressed as

Eq. 2.32 and Eq. 2.33.

$$h_\theta(f_k) = \frac{1}{1+e^{-\theta^T f_k}} \tag{2.32}$$

$$P_{k,yield} = 1, \text{ if } \; h_\theta(f_k) > \beta_{yield} \tag{2.33}$$

where $f_k = [1, \phi_k, d_0, v_0, s_k - s_0, v_k, v_{k,front}]$ is the feature describing the current state, and $\beta_{yield} = 0.85$ is a threshold. Using this yielding classifier, the ego car is able to take its interaction with the human car into account when generating the action for the next time step.

Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) is an efficient search technique, which can balance the trade-off between exploration and exploitation. When we use a tree search, it is very likely that the current perceived best action is not the global optimal action. MCTS is good at dealing with this problem. It is able to search the unexplored nodes in the tree while also be favor of the area where better actions exist. Therefore, MCTS can result in relatively good action without a traverse of the tree. Actually, as the number of iterations increase, the result of MCTS converges to the real optimal action in terms of probability [75]. This property can reduce the computational burden when solving our POMDP framework, and generate a best action for the autonomous vehicle to execute at the next time step within a relatively small time period. The flow chart of MCTS we used in the algorithm is shown in figure 2.1.

When we implement MCTS, we need to do rollout over a lookahead horizon. In the process of rollout, we have to predict the actions of other human cars, so that we can get the expected reward over the 10 seconds horizon. As aforementioned, we use Velocity Driver Model (VDM) to generate the motion of other cars in the prediction module when we do the rollout. The yielding classifier first takes in the feature and outputs a probability of cooperative behavior of the other car. Then if the probability of yielding to the ego car is bigger than the threshold, the front car in VDM is set to be the ego car, and the parameters of VDM are set to be those learned from successful merging scenarios. If the probability of yielding to the ego car is less than the threshold, then the front car in VDM is set to be the lead human car on the target lane, and the parameters of VDM are set to be those learned from unsuccessful merging scenarios. This setting makes sense because the behavior patterns of a human car would be different between the cooperative and uncooperative scenarios. In this way, the ego car is aware of the intention of the other human cars, and is able to make decisions in consideration of its interaction with other cars.

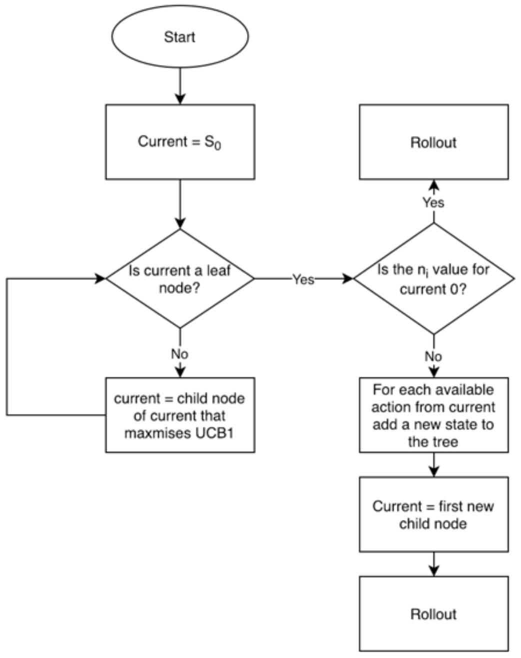

Start
Current = S0
Is current a leaf node?
Yes
No
current = child node of current that maxmises UCB1
Is the ni value for current 0?
Yes
Rollout
No
For each available action from current add a new state to the tree
Current = first new child node
Rollout

Figure 2.1: MCTS Flow Chart

2.4 Experiment Results

We implemented the distribution for the parameters found using MLE into an POMDP described in the Problem Setting Section. We used MATLAB to carry out the experiments on a system with a 2.9GHz Intel Core i7-7500U CPU.

Parameter Estimation Results

The results of the IDM parameter estimation for successful trials are shown in Figs. 2.2. A histogram of each parameter is shown on the left axis, where the frequency is plotted against the parameter values. We compute the mean and standard deviation of the parameters for the trials, and the corresponding normal distribution is given on the second right axis. We assume the parameters are independently distributed. The assumed prior distribution in Table 2.1 used in by the MAP estimation is given. As expected, the distribution calculated from the MAP estimation tends to more closely follow a normal distribution. This is particularly apparent in the estimate reference velocity, v_0, where the parameters estimated using ML are quite varied. Conversely, with the prior distribution for the MAP estimation, the estimated parameters are closer to the expected mean of 20 m/s. The power factor, δ, is also quite varied when using ML, but the prior estimate with MAP reduces the variation in the estimate values of δ. Since the typical values for the IDM parameters are well-known from prior literature

Estimation Method	**Model**	**Average MSE**	**Max MSE**
ML	SDM	1167.66	4948.86
	VDM	0.046	0.139
	IDM	0.111	0.610
	IDM w/ time shift	0.072	0.451
MAP	SDM	15.13	35.03
	VDM	0.054	0.159
	IDM	0.121	0.467
	IDM w/ time shift	0.073	0.451

Table 2.2: Average and maximum (over all trials) mean-squared error for each parameter estimation method

[137], the MAP method provides a consistent solution that reduces the susceptibility of the estimation to noise.

Next, we compare the accuracy of the predicted values using the various driver models: IDM, VDM, and SDM. We performed the estimation using LS, ML, and MAP for each of the driver models. Table 2.2 gives the mean square error (MSE) between the measured acceleration and predicted acceleration statistics for all of the trials for the ML and MAP methods (LS is excluded since the formulation is the same as ML). The average and maximum MSE over all the trials is the least for the VDM, following by the IDM. For most of the trials the trends in the IDM seemed to be delayed from the measured states. Therefore we introduced an additional term t_{shift} to delay the calculation of the driver model to depend on the previous states so that $a_{pred}(t) = f(x(t - t_{shift}))$. The IDM with the delay introduced by the time shift of 1s is reduced the MSE compared to the IDM without an introduced delay. The delay was introduced to test the possibility that there is a delay in the driver's observation of the current state. However, the VDM still performed better than the IDM even after introducing the time delay. The SDM performed significantly worse than both the IDM and VDM, and therefore was determined to not accurately predict the behavior of other cars during the merge maneuver.

The estimated means and covariance matrix for trials with a successful merge are given in Table 2.3 – 2.6 using ML and MAP respectively. The parameters estimated using MAP have smaller standard deviations than those found using ML. Additionally, the estimated parameters found by MAP are closer to the assumed prior parameter values than those found using ML. The additional likelihood term in the cost function in the MAP formulation adds cost for deviation from the assumed parameters as priors, which reduces the variation in the parameter estimation. Here, we empirically prove that MAP can integrate the information from the prior distribution into parameter fitting. However, the assumed prior parameter distribution is not necessary the optimal one since there is no ground truth values. Therefore, we choose to use the parameter estimation results by MLE to do simulations in the evaluation section.

Model	**Parameters θ**	**Mean μ_0**	**Covariance Σ_0**
IDM	$\begin{bmatrix} T \\ a_{max} \\ v_0 \\ \delta \\ d_0 \\ b \end{bmatrix}$	$\begin{bmatrix} 0.522 \\ 0.837 \\ 13.257 \\ 5.696 \\ 4.543 \\ 0.805 \end{bmatrix}$	$\text{diag}\left(\begin{bmatrix} 0.710 \\ 0.691 \\ 13.484 \\ 7.990 \\ 3.776 \\ 1.476 \end{bmatrix}\right)$
VDM	$\begin{bmatrix} V_1 \\ V_2 \\ C_1 \\ C_2 \\ \lambda \\ \kappa \end{bmatrix}$	$\begin{bmatrix} 4.760 \\ 5.158 \\ 1.748 \\ 3.386 \\ 1.455 \\ 0.476 \end{bmatrix}$	$\text{diag}\left(\begin{bmatrix} 3.293 \\ 3.39 \\ 1.945 \\ 3.389 \\ 2.136 \\ 0.950 \end{bmatrix}\right)$

Table 2.3: Maximum Likelihoood parameter estimation results for trials with successful merge

Model	**Parameters θ**	**Mean μ_0**	**Covariance Σ_0**
IDM	$\begin{bmatrix} T \\ a_{max} \\ v_0 \\ \delta \\ d_0 \\ b \end{bmatrix}$	$\begin{bmatrix} 0.958 \\ 1.421 \\ 16.885 \\ 3.426 \\ 1.281 \\ 61.907 \end{bmatrix}$	$\text{diag}\left(\begin{bmatrix} 1.995 \\ 0.769 \\ 15.430 \\ 2.191 \\ 2.484 \\ 483.360 \end{bmatrix}\right)$
VDM	$\begin{bmatrix} V_1 \\ V_2 \\ C_1 \\ C_2 \\ \lambda \\ \kappa \end{bmatrix}$	$\begin{bmatrix} 3.747 \\ 6.133 \\ 1.641 \\ 7.1181 \\ 0.530 \\ 0.332 \end{bmatrix}$	$\text{diag}\left(\begin{bmatrix} 3.507 \\ 3.433 \\ 2.289 \\ 3.528 \\ 0.514 \\ 0.388 \end{bmatrix}\right)$

Table 2.4: Maximum Likelihood parameter estimation results for trials with unsuccessful merge

Simulation Results

In the experiment, we set the horizon to be 12 seconds. There were totally three cars: one was the ego car in lane 1, intending to merge into lane 0, and the other two cars were the other human cars in lane 0. The ego car used POMDP to obtain its action for the next time step. The time step was 1 second. When solving MCTS for POMDP, we used IDM/VDM and a yield classifier to predict the motion of the other cars. IDM was a

Model	**Parameters θ**	**Mean μ_0**	**Covariance Σ_0**
IDM	$\begin{bmatrix} T \\ a_{max} \\ v_0 \\ \delta \\ d_0 \\ b \end{bmatrix}$	$\begin{bmatrix} 0.447 \\ 0.598 \\ 16.66 \\ 5.92 \\ 3.42 \\ 1.43 \end{bmatrix}$	$\text{diag}\left(\begin{bmatrix} 0.403 \\ 0.522 \\ 7.044 \\ 5.187 \\ 2.432 \\ 1.298 \end{bmatrix}\right)$
VDM	$\begin{bmatrix} V_1 \\ V_2 \\ C_1 \\ C_2 \\ \lambda \\ \kappa \end{bmatrix}$	$\begin{bmatrix} 5.879 \\ 7.300 \\ 0.466 \\ 1.078 \\ 0.622 \\ 0.268 \end{bmatrix}$	$\text{diag}\left(\begin{bmatrix} 1.894 \\ 1.510 \\ 0.370 \\ 0.866 \\ 0.476 \\ 0.303 \end{bmatrix}\right)$

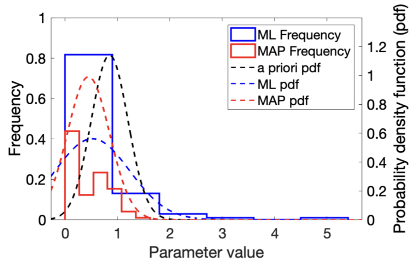
Table 2.5: Maximum a Posteriori parameter estimation results for trials with successful merge

Model	**Parameters θ**	**Mean μ_0**	**Covariance Σ_0**
IDM	$\begin{bmatrix} T \\ a_{max} \\ v_0 \\ \delta \\ d_0 \\ b \end{bmatrix}$	$\begin{bmatrix} 1.468 \\ 0.853 \\ 19.15 \\ 4.047 \\ 1.186 \\ 3.579 \end{bmatrix}$	$\text{diag}\left(\begin{bmatrix} 0.522 \\ 0.549 \\ 3.849 \\ 0.094 \\ 1.235 \\ 0.943 \end{bmatrix}\right)$
VDM	$\begin{bmatrix} V_1 \\ V_2 \\ C_1 \\ C_2 \\ \lambda \\ \kappa \end{bmatrix}$	$\begin{bmatrix} 5.650 \\ 8.143 \\ 0.953 \\ 2.650 \\ 0.546 \\ 0.289 \end{bmatrix}$	$\text{diag}\left(\begin{bmatrix} 1.354 \\ 0.994 \\ 1.930 \\ 1.363 \\ 0.367 \\ 0.253 \end{bmatrix}\right)$

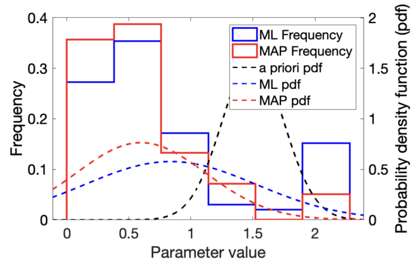
Table 2.6: Maximum a Posteriori parameter estimation results for trials with unsuccessful merge

baseline whose parameters were set according to widely accepted expert data from previous literatures. Before starting each trial, we sampled parameters for VDM from the learned Gaussian distribution of successful and unsuccessful merge. The solver had a 10-second look-ahead to search for the actions with the maximum reward. Then the ego car executed the best action returned from the search for the next time step.

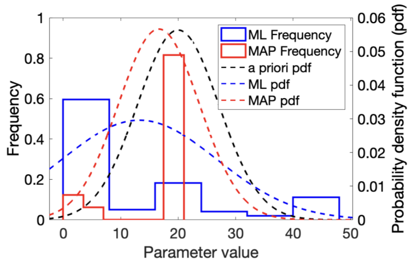
Those two other human cars were assumed to be able to change their longitudinal acceler-



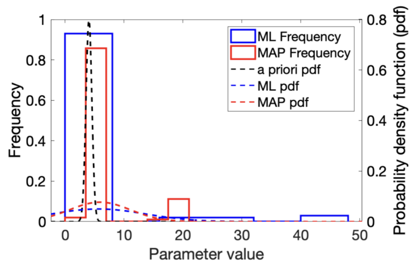
(a) Minimum following time, T



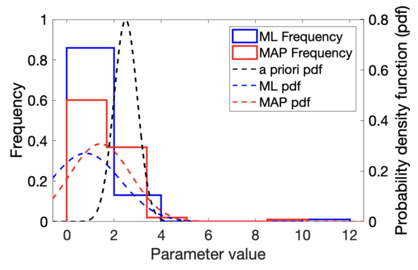
(b) Maximum acceleration, a_{max}



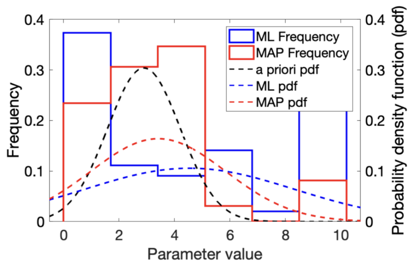
(c) Reference velocity, v_0



(d) Power factor, δ



(e) Minimum following distance, d_0



(f) Braking deceleration, b

Figure 2.2: IDM estimation results of successful merge trials. Frequency of parameter estimation values among trials (left axis) and probability density function (pdf) of a prior distribution and estimated parameters distributions (right axis)

ation only, staying at the center line of lane 0. We used the VDM with sampled parameters from the learned distribution by MLE to steer the other cars in the test environment to interact with the ego car.

We compared the performance of using IDM or VDM as the predictor in the rollout based on 500 trials for each. The ego car can successfully merge into the target lane with a rate of 99.4% when using the VDM to predict the other cars' motion. However, the performance of original IDM as a predicted model for the other cars was not as good, with a success rate as 87.4%. Besides, the error of the predicted acceleration, velocity and position of the other cars were also small when using VDM, indicating that the VDM with a set of tuned parameters can better describe the motion of the other cars in the real world. Those results are shown in Table 2.7.

In the following paragraph, two trials of merging are discussed: one is in a scenario where the rear human car N_1 yielded to the ego agent, while in the other scenario N_1 did not yield. These two trials began with the same initial conditions and are implemented in a time horizon of $t = 12s$. The ego car was able to recognize the intentions of the human car using the yield classifier and the tuned prediction model, and executed the lane changing task successfully whether or not the human car yielded.

In Figure 2.3, we show a trial in which the rear human car yielded to the ego car. The position and the velocity of the three cars are shown in Figure 2.5. As the figures show, N_1 in lane 0 decelerated when the ego car cut in, indicating that they were interacting with

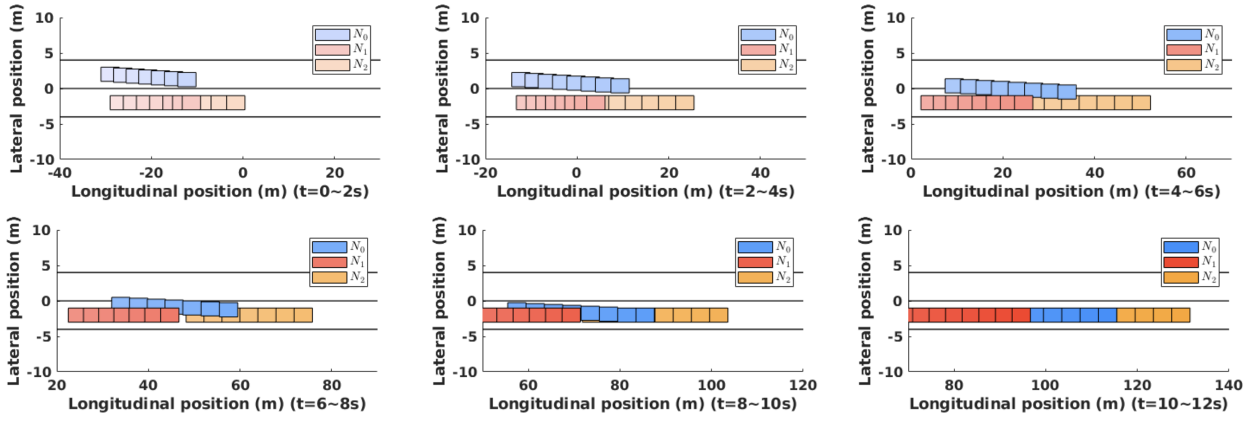


Figure 2.3: The trajectory of the ego agent when the back human car is yielding to it. The continuous trajectory is sliced into 6 pieces for clarity. Each trajectory piece lasts 2 second and the whole trajectory lasts 12 seconds.

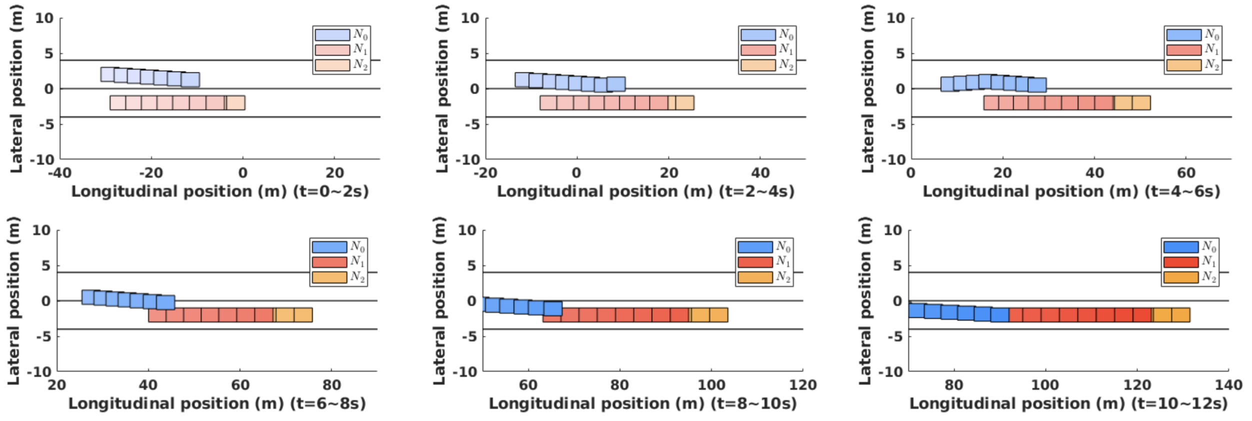


Figure 2.4: The trajectory of the ego agent when the back human car is not yielding to it. The continuous trajectory is sliced into 6 pieces for clarity. Each trajectory piece lasts 2 second and the whole trajectory lasts 12 seconds.

each other. Meanwhile, the other human car N_2 was accelerating to the reference velocity $16m/s$ as it was the speed limit. The ego car was able to find the right time point (around $t = 2s$) to execute the merging attempt and then can successfully merge into the target lane as shown in Figure 2.3. In Figure 2.4, we show a trial in which the rear human car did not yield to the ego car. The position and the velocity of the three cars are shown in Figure 2.6. As shown in the velocity plot in Figure 2.6, the ego car N_0 accelerated and attempted to merge at around $t = 0\ 1s$. However, N_1 also accelerated, which means that it did not intend to yield to the ego car N_0. Thus, the ego car N_0 decelerated a little bit, and then merged into the target lane, behind those two human cars N_1 and N_2. After successfully merging

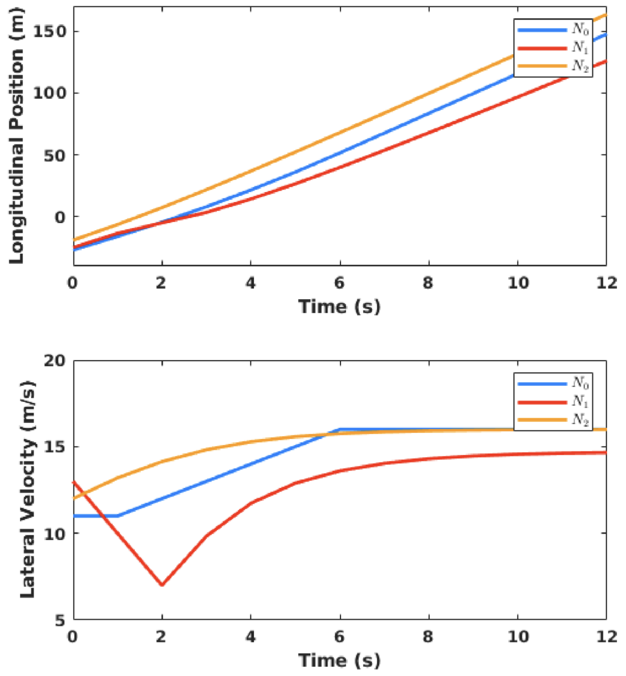


Figure 2.5: The longitudinal position and velocity of the cars when the back human car is yielding to the ego agent.

into the target lane, the ego car N_0 then accelerated to the reference speed $v_{ref} = 16m/s$, and followed those two human cars afterwards.

Real-World Dataset Results

We utilized the INTERACTION dataset [156] to validate the proposed algorithms. Similar to the setup in the calibration implementation, we selected the motion data of similar merging scenarios from the dataset, and used the selected data as the validation dataset. Each scenario in the validation set contains the initial position and trajectories of the ego car, as well as trajectories of two other human cars that are driving on the lane adjacent to the initial lane of the ego car. The goal of the ego car was to merge to the lane with two other cars. The scenarios of the real-traffic dataset were very similar to the simulation in the previous section. We selected 200 merging scenarios from the original dataset, and perform lane change behavior planning in each of them, which is similar to the tasks in the simulation. The statistical results are shown in table 2.7. We define a ”success” in behavior planning as that a collision-free trajectory is generated. Specifically it means that there is no intersection between the ego car’s trajectory and the other cars’ trajectories at each time step. There were 190 collision free merging attempts out of 200 planning trials, with an average error of 0.606 m/s^2 in acceleration prediction, an average error of $1.783m/s$ in velocity prediction,

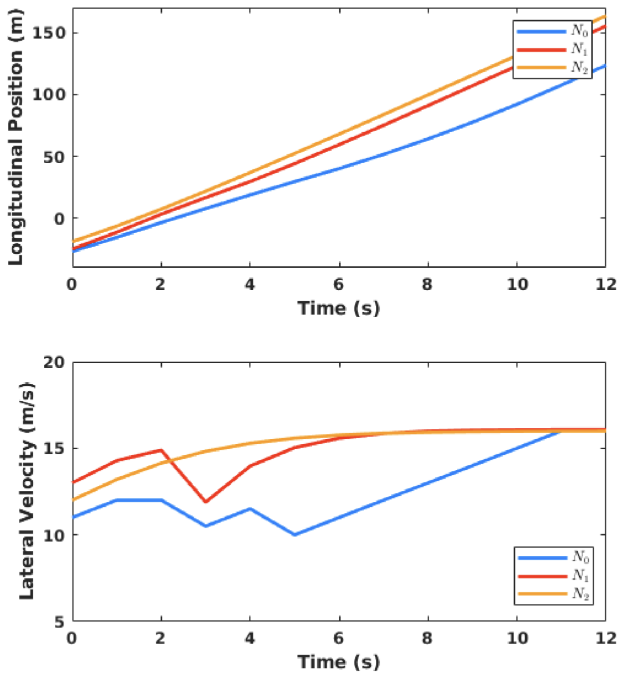
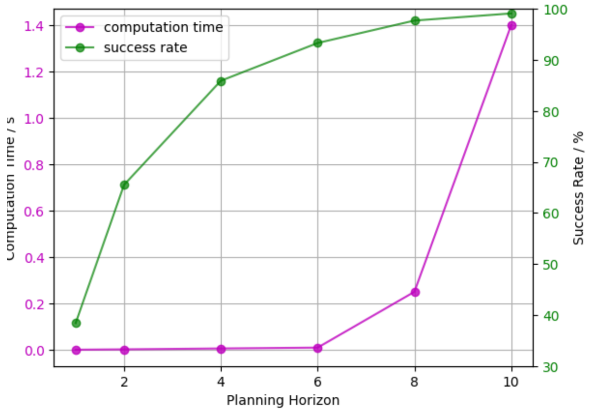


Figure 2.6: The longitudinal position and velocity of the cars when the back human car is not yielding to the ego agent.

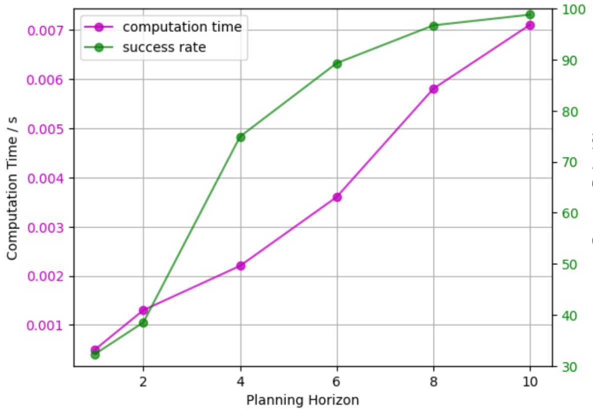
Table 2.7: The prediction error and the success merges of IDM/VDM.

SIMULATION				
model	**a error**	**v error**	**x error**	**success**
IDM	0.610 m/s^2	1.261 m/s	0.630 m	437/500
VDM	0.483 m/s^2	1.134 m/s	0.535 m	497/500
REAL TRAFFIC DATA				
model	**a error**	**v error**	**x error**	**success**
VDM	0.606 m/s^2	1.783 m/s	0.891 m	190/200

and an average error of $0.891m$ in position prediction. All performance indices including a success rate, were lower than those in simulation, which indicates it is harder to predict the motion of a real human car. However, the proposed planner was still able to perform a merging maneuver with a relatively high success rate in real-world scenarios.



(a) Brute Force



(b) MCTS

Figure 2.7: The computation time and success rate comparison between (a) brute force and (b) MCTS.

MCTS vs. Brute Force

MCTS reduced the computation time of solving the POMDP for the planning problem to a large extent. We aim to further elucidate the advantage of applying MCTS by comparing it to brute force solutions in this section. The ablation study is done in the same simulator as Sec. 2.4, with various planning horizon from 1 to 10 steps. Specifically, we compare the success rate and computation time between the brute force solution and the MCTS solution of the POMDP.

As shown in Fig. 2.7 (a), the brute force solution has near-optimal success rate, but the computation time explodes as the horizon increases from 1 to 10 steps. It is no longer practical to operate the brute force solver when the lookahead horizon is larger than 6s, as the computation time is longer than the planning interval 0.1s. The solver cannot output a solution in time for each time step.

The MCTS solver, however, can attain satisfactory performance within reasonable computation time at around 0.007s for 10 steps lookahead. The MCTS computation time benefits largely from the pruning process and the heuristic searching for a solution. The success rate increases from a disastrous 30% to a near-perfect 98% as the planning horizon changes from 1 to 10 steps, which indicates the MCTS benefits from the longer lookahead horizon. The gap between the success rate of MCTS and brute force remains small at no more than 3.5% when the lookahead horizon is larger than 6 steps, while MCTS keeps the computation time reasonable for a 10Hz planning frequency.

2.5 Conclusion

In this report, we presented a POMDP framework to deal with behavior planning in a lane changing scenario, while considering the interaction between the autonomous car and the surrounding human cars. In the planning module, a yielding classifier was trained by a real-world dataset, predicting intentions and behavior of the human cars by switching the driver model. The switching process was based on the current and historical states, which indicates how likely the human car will drive cooperatively. The driver model switched between two sets of parameters for successful and unsuccessful merges, respectively, learned from real-world data. Evaluations on simulation showed that the proposed framework with learned parameters is able to perform safe and efficient behavior planning for autonomous cars in the lane changing scenarios. We also validated our planner with real-world traffic data environment. The proposed planner with refined prediction model outperformed the planner with a fixed-parameter driver model extracted.

The driver models in our planner are extracted from real-world datasets. We explored parameter estimation of various driver models in order to more accurately predict the motion of other cars. Least squares, maximum likelihood, and maximum a posteriori were used to estimate the parameters of the driver models. Of the three driver models were investigated (IDM, SDM, and VDM), the VDM had the least prediction error after parameter estimation, followed by the IDM. The SDM had significantly worse prediction error, and therefore we concluded that this model is not an accurate predictor of the state of the other cars for the data collected from traffic merges. The MDP formulation used the prior known distribution collected from other works that explored the parameter values. This prior distribution stabilized the results of the parameter estimation over the multiple trials. The resulting distribution of the parameters captures the variability in driver behavior. We demonstrate the usefulness of the found parameter distribution by implementing a sampling from the found distribution to find the parameters in the transition model of an POMDP describing the ego car merging onto a highway.

While we account for a scenario where a car successfully merges ahead of the car in question and for instances where the merge was unsuccessful, the data was collected from limited locations with certain types of merging maneuver. We identify potential avenues for future work:

- Expand sampled data to include different locations and/or types of merging maneuvers
- Application of the parameter estimation method into other driving scenarios (e.g. intersections) and other data sources
- Online parameter estimation or adaptive control to identify a vehicle's parameters in real time
- Incorporate parameter distribution into a stochastic transition model for the other cars
- Incorporate various persuasion model to the POMDP framework

This work provides a basis for understanding how a distribution for driver model parameters, instead of static fixed parameters can be useful for capturing the variability in driver behavior. Future work can expand on this idea to incorporate different driver behavior into prediction models that will better help autonomous cars plan how to interact with their environment.

Chapter 3

Dealing with the Unknown: Pessimistic Offline Reinforcement Learning

Reinforcement Learning (RL) has been shown effective in domains where the agent can learn policies by actively interacting with its operating environment. However, if we change the RL scheme to offline setting where the agent can only update its policy via static datasets, one of the major issues in offline reinforcement learning emerges, i.e. distributional shift. We propose a Pessimistic Offline Reinforcement Learning (PessORL) algorithm to actively lead the agent back to the area where it is familiar by manipulating the value function. We focus on problems caused by out-of-distribution (OOD) states, and deliberately penalize high values at states that are absent in the training dataset, so that the learned pessimistic value function lower bounds the true value anywhere within the state space. We evaluate the PessORL algorithm on various benchmark tasks, where we show that our method gains better performance by explicitly handling OOD states, when compared to those methods merely considering OOD actions.

3.1 Introduction

Reinforcement learning (RL), especially with high-capacity models such as deep nets, has shown its power in many domains, e.g., gaming, healthcare, and robotics. However, typical training schemes of RL algorithms rely on active interaction with the environments. It limits their applications in domains where active data collection is expensive or dangerous (e.g., autonomous driving). Recently, offline reinforcement learning (offline RL) has emerged as a promising candidate to overcome this barrier. Different from traditional RL methods, offline-RL learns the policy from a static offline dataset collected without iterative interaction with the environment. Recent works have shown its ability in solving various policy learning tasks [62, 67, 66]. However, offline RL methods suffer from several major problems. One of them is distributional shift. Unlike online RL algorithms, the state and action distributions are different during training and testing. As a result, RL agents may fail dramatically after deployed online. For example, in safety-critical applications such as autonomous driving, overconfident and catastrophic extrapolations may occur in out-of-distribution (OOD) scenes [43].

Many prior works [74, 71, 102, 42, 3, 101] try to mitigate this problem by handling OOD actions. They discourage the policies to visit OOD actions by designing conservative value functions, or estimating the uncertainty of Q-functions. Although constraining the policy can implicitly mitigate the problem of state distributional shift, few works have adopted measures to explicitly handle OOD states during the training stage. In this work, we propose the Pessimistic Offline Reinforcement Learning (PessORL) framework to explicitly limit the policy from visiting both unseen states and actions. We refer to the states or the actions that are not included in the training data as the unseen states or the unseen actions.

Our PessORL framework is inspired by the concept of pessimistic MDP in [68], where the reward is significantly small for unseen state-action pairs. We aim to limit the magnitude of the value function at unseen states, so that the agent can avoid or recover from unseen states. It is then crucial to precisely detect OOD states and shape the value function at those states. Since prior methods on OOD actions are derived from a similar concept, we can adapt their approaches to handle OOD states. There are mainly two approaches in the literature. One is to estimate the epistemic uncertainty of Q-function and subtract it from the original Q-function to get a conservative Q-function [71, 102, 42, 3, 101]. The other is to regularize the Q-function during the learning process [74]. The first method is highly sensitive to the trade-off between the uncertainty estimation and the original Q-function [147, 159] and the quality of uncertainty estimation [78].

Therefore, we follow the second approach, and add a conservative regularization term to the policy evaluation step of PessORL to shape the value function. We prove that PessORL learns a pessimistic value function that lower bounds the true value function, and forces the policy to avoid or recover from out-of-distribution states and actions. We evaluate the PessORL algorithm on various benchmark tasks. The performance of our method matches the state-of-the-art offline RL methods. In particular, we show that, by explicitly handling OOD states, we can further improve the policy performance compared to those methods merely considering OOD actions.

3.2 Related Work

A big challenge for offline reinforcement learning methods is to deal with the problems caused by unvisited states or actions in training data, which is also known as distributional shift. In model-free offline reinforcement learning, some works used importance sampling to fill the gap between the learned policy and the behavior policy in the training dataset [110, 48, 56, 105, 27]. There are also many works constrained the learned policy to be similar to the behavior policy by explicit constraints in the training dataset [71, 46, 127, 108], so that the agent can avoid out-of-distribution actions during test time. The work in [159] proposed a latent space to constrain the policy to avoid deviating from the training data support. One further step to make the agent avoid actions that may cause itself deviate from the training data support is to get a conservative value function and thus a conservative policy. The works in [102, 42, 71, 3, 101] estimate the uncertainty of the learned Q-function, and then directly subtract it from the Q-function to get a conservative Q-function. Another way to get a conservative Q-function is to regularize the Q-function in the optimization problem during the learning process [74]. In model-based reinforcement learning (MBRL), there are also many algorithms that constrain the exploitation in the environment with effective uncertainty estimation methods [134, 146, 157, 52, 60, 151, 152, 68]. It is considered to be mature and reliable to detect OOD actions and states by methods from MBRL. Most of the aforementioned methods focus on OOD actions but not have explicit mechanism to deal with OOD states. In this chapter, we focus on OOD states and propose a method to learn a pessimistic value function by adding regularization terms when updating Q-functions, and follow the works in the MBRL domain to establish the module to detect OOD states in our algorithm.

3.3 Background

Offline Reinforcement Learning

Given a Markov decision process (MDP), an RL agent aims to maximize the expectation of cumulative rewards. The MDP is represented by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and γ is the discount factor. Typical RL algorithms optimize the policy using experience collected when interacting with the environment. Unlike those online learning paradigms, offline-RL algorithms rely solely on a static offline dataset, denoted by $\mathcal{D} = \left\{ \left(\mathbf{s}_t^i, \mathbf{a}_t^i, \mathbf{s}_{t+1}^i, r_t^i \right) \right\}$.

In this work, we focus on dynamic-programming-based RL algorithms under the offline setting, where we extract a policy from a learned value function for the underlying MDP in the training data. Standard Q-learning method estimates an approximate Q-function parametrized by θ, i.e. $\hat{Q}_\theta(\mathbf{s}_t, \mathbf{a}_t)$. In each iteration, the Q-function is updated as follows:

$$\hat{Q}_\theta^{k+1} \leftarrow \arg\min_{Q} \frac{1}{2} \mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}' \sim \mathcal{D}} \left[\left(Q(\mathbf{s}, \mathbf{a}) - \hat{\mathcal{B}}^\pi \hat{Q}_\theta^k(\mathbf{s}, \mathbf{a}) \right)^2 \right] \text{ (policy evaluation)}, \tag{3.1}$$

where $\hat{\mathcal{B}}^\pi$ is the empirical Bellman update operator defined as:

$$\hat{\mathcal{B}}^\pi \hat{Q}_\theta^k(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{a}' \sim \hat{\pi}^k(\mathbf{a}'|\mathbf{s}')} \left[\hat{Q}_\theta^k(\mathbf{s}', \mathbf{a}') \right]. \tag{3.2}$$

For discrete action space, we define $\hat{\pi}^k$ as the optimal policy induced by the learned Q-function, i.e. $\hat{\pi}^k(\mathbf{a}|\mathbf{s}) = \delta \left[\mathbf{a} = \arg\max_{\mathbf{a}} \hat{Q}_\theta^k(\mathbf{s}, \mathbf{a}) \right]$. In this case, $\hat{\mathcal{B}}^\pi$ collides into the Bellman optimality operator. When the action space is continuous, we follow actor-critic algorithms to approximate the optimal policy by executing a policy improvement step after policy evaluation in each iteration:

$$\hat{\pi}^{k+1} \leftarrow \arg\max_{\pi} \mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[\hat{Q}_{\theta}^{k+1}(\mathbf{s},\mathbf{a})\right] \text{ (policy improvement).} \tag{3.3}$$

In the rest of the chapter, we denote $\mathcal{E}(Q, \hat{\mathcal{B}}^{\pi}\hat{Q}_{\theta}^{k}) = \frac{1}{2}\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}}\left[\left(Q(\mathbf{s},\mathbf{a}) - \hat{\mathcal{B}}^{\pi}\hat{Q}_{\theta}^{k}(\mathbf{s},\mathbf{a})\right)^2\right]$ as the Bellman update error for simplicity.

Uncertainty-Based Methods and Pessimistic Value Functions

By observing Eqn. 3.1, it is obvious that the Q-function $\hat{Q}_{\theta}$ is never evaluated or updated at states or actions that never appear in the dataset. The agent may behave unexpectedly or unpredictably at those unseen states or actions during test time. For dynamic-programming-based approaches, one way to address the issue of unseen actions is to estimate the epistemic uncertainty of Q-function and subtract it from the original Q-function [71, 102, 42, 3, 101]. The uncertainty is estimated based on an ensemble of learned Q-functions, and the final conservative Q-function becomes $Q_{\mathrm{c}}(\mathbf{s},\mathbf{a}) = \mathbb{E}_{Q\sim\mathcal{P}_{\mathcal{D}}(Q)}[Q(\mathbf{s},\mathbf{a}) - \alpha \mathrm{Unc}(\mathcal{P}_{\mathcal{D}}(Q))]$, where Unc is defined to be some uncertainty estimation metric, and $\mathcal{P}_{\mathcal{D}}(Q)$ is the distribution over possible Q-functions. Because the uncertainty metric is directly subtracted, uncertainty-based methods is highly sensitive to the quality of uncertainty estimation. Meanwhile, it is difficult to find an ideal α to balance the original Q-function and Unc.

Another way is to regularize the Q-function at the step of policy evaluation. A representative example is Conservative Q-Learning (CQL) [74]. Assuming that the dataset $\mathcal{D}$ is collected with a behavior policy $\pi_{\beta}(\mathbf{a}|\mathbf{s})$, and $\hat{\pi}^{k}(\mathbf{a}|\mathbf{s})$ is the learned policy at iteration k, the policy evaluation step in CQL becomes:

$$\hat{Q}^{k+1} \leftarrow \arg\min_{Q} \alpha\left(\mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\hat{\pi}^{k}(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] - \mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi_{\beta}(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})]\right) + \mathcal{E}(Q, \hat{\mathcal{B}}^{\pi}\hat{Q}_{\theta}^{k}). \tag{3.4}$$

In the rest of the chapter, we denote $\mathcal{C}(Q)$ as the cost term adopted from the CQL, i.e., $\mathcal{C}(Q) = \alpha\left(\mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\hat{\pi}^{k}(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] - \mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi_{\beta}(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})]\right)$. It is worth noting that the aforementioned methods all focus on OOD actions, but they do not have an explicit mechanism to deal with OOD states, which motivates us to develop the PessORL framework in this work.

3.4 Pessimistic Offline Reinforcement Learning Framework

In this section, we introduce the PessORL framework to mitigate the issue of state distributional shift. In particular, we propose a novel conservative regularization term in the policy evaluation step. It can then be integrated into Q-learning or actor-critic algorithm, which will be described in Sec. 3.5.

How To Deal With OOD States

Assuming the dataset $\mathcal{D}$ is collected with a behavior policy $\pi_\beta(\mathbf{a}|\mathbf{s})$, and the states $\mathbf{s}$ are distributed according to a distribution $d^{\pi_\beta}(\mathbf{s})$ in the dataset, we propose to solve the problem caused by state distributional shift by augmenting the policy evaluation step in CQL [74] with a regularization term scaled by a trade-off factor ε:

$$\min_Q \varepsilon \left(\mathbb{E}_{\mathbf{s}\sim d^\phi(\mathbf{s}),\mathbf{a}\sim\hat{\pi}^k(\mathbf{a}|\mathbf{s})}\left[Q(\mathbf{s},\mathbf{a})\right] - \mathbb{E}_{\mathbf{s}\sim d^{\pi_\beta}(\mathbf{s}),\mathbf{a}\sim\hat{\pi}^k(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})]\right) + \mathcal{E}(Q, \hat{\mathcal{B}}^\pi \hat{Q}^k_\theta) + \mathcal{C}(Q), \quad (3.5)$$

where $d^\phi(\mathbf{s})$ is a particular state distribution of our choice.

The idea is to use the minimization term $\varepsilon\left(\mathbb{E}_{\mathbf{s}\sim d^\phi(\mathbf{s}),\mathbf{a}\sim\hat{\pi}^k(\mathbf{a}|\mathbf{s})}\left[Q(\mathbf{s},\mathbf{a})\right]\right)$ to penalize high values at unseen states in the dataset, and the maximization term $\varepsilon\left(\mathbb{E}_{\mathbf{s}\sim d^{\pi_\beta}(\mathbf{s}),\mathbf{a}\sim\hat{\pi}^k(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})]\right)$ to cancel the penalization at in-distribution states. The regularized Q-function could then push the agent towards regions close to the states from the dataset, where the values are higher. To achieve it, we need to find a distribution $d^\phi(\mathbf{s})$ that assigns high probabilities to states far away from the dataset, and low probabilities to states near the training dataset. We will instantiate a practical design of $d^\phi(\mathbf{s})$ in Sec. 3.5. For now, we just assume $d^\phi(\mathbf{s})$ assigns high probabilities to OOD states.

Theoretical Analysis

In this section, we analyze the theoretical properties of the proposed policy evaluation step. The proof and more details can be found in Appendix 3.4.

We define $k \in \mathbb{N}$ as the iteration of policy evaluation, i.e. $\hat{Q}^k$ denotes the optimized Q-function in the $k-$th iteration obtained by PessORL. Q^π is defined to be the true Q-function under a policy $\pi(\mathbf{a}|\mathbf{s})$ in the underlying MDP without any regularization. The true Q-function can be written in a recursive form via the exact Bellman operator, $\mathcal{B}^\pi$, as $Q^\pi = \mathcal{B}^\pi Q^\pi$. We define $\hat{V}^k$ as the value function under a policy $\pi(\mathbf{a}|\mathbf{s})$, $\hat{V}^k(\mathbf{s}) = \mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}[\hat{Q}^k(\mathbf{s},\mathbf{a})]$. For the true value function V^π in the underlying MDP, we also have $V^\pi = \mathcal{B}^\pi V^\pi$.

We first introduce the theorem that the learned value function is a lower bound of the true one without considering the sampling error defined in the Lemma 3.4.1.

Jibberish 1 *Assume we can obtain the exact reward function $r(\mathbf{s},\mathbf{a})$ and the transition function $T(\mathbf{s}'|\mathbf{s},\mathbf{a})$ of the underlying MDP. Let $\hat{V}^\pi(\mathbf{s}) = \lim_{k\to\infty}\hat{V}^k(\mathbf{s})$. Then $\forall \mathbf{s} \in \mathcal{S}$, the learned value function via Eqn. 3.5 is a lower bound of the true one, i.e., $\hat{V}^\pi(\mathbf{s}) \leq V^\pi(\mathbf{s})$, if the ratio $\dfrac{\varepsilon}{\alpha}$ satisfies*

$$\frac{\varepsilon}{\alpha} \leq \min_{\mathbf{s}} \left(\sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s})\left[\frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right]\right)\left(\frac{\left|d^\phi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s})\right|}{d^{\pi_\beta}(\mathbf{s})}\sum_{\mathbf{a}}\frac{\pi^2(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})}\right)^{-1}.$$

It is worth noting that the learned value function still lower bounds the true value function for any state and action in the training datasets, i.e. $\mathbf{s}, \mathbf{a} \in \mathcal{D}$, even when we consider the sampling error defined in the Lemma 3.4.1. Further details are shown in Corollary 3.4.1. We have no reward or transition pair collected at unseen states or actions outside the training dataset, so it is impossible to bound the error outside the training dataset when consider the sampling error introduced by the reward function and the transition function.

We can now step further and show that the values at OOD states are lower than those at in-distribution states based on the learned value function. The proof can be found in Appendix 3.4.

Jibberish 2 *For any state $\mathbf{s} \in \mathcal{S}$, if $\varepsilon > 0$ is sufficiently large, then the expectation of the learned value function via Eqn. 3.5 under the state marginal $d^{\pi_\beta}(\mathbf{s})$ in the training data is higher than that under $d^{\phi}(\mathbf{s})$, i.e., $\mathbb{E}_{\mathbf{s}\sim d^{\phi}(\mathbf{s})}[\hat{V}^{\pi}(\mathbf{s})] < \mathbb{E}_{\mathbf{s}\sim d^{\pi_\beta}(\mathbf{s})}[\hat{V}^{\pi}(\mathbf{s})]$.*

During training time, we can at least evaluate Q-values of OOD actions based on in-distribution states. However, there is actually no information about immediate rewards at OOD states, thus no information about Q-values. Intuitively, under offline settings, the best we can do to mitigate the problem of OOD states is to suppress values at these OOD states, and raising values at in-distribution states, so that the agent can be attracted to the area where it is familiar near the training data. Thm. 2 indeed tells us PessORL models a value function that assigns smaller values to OOD states compared to those at in-distribution states. Optimizing a policy under such a value function is similar to forcing the policy to avoid unknown states and actions.

In summary, PessORL can learn a pessimistic value function that lower bounds the true value function. Furthermore, this value function assigns smaller values to OOD states compared to those at in-distribution states, which helps the agent avoid or even recover from OOD states.

Proof of Theorem

In this section, we provide the proofs of the theorems in this chapter.

Remark. We provide the proofs under a tabular setting. Most continuous space can be approximately discretized to a tabular form, although the cordiality of the tabular form may be large. We define P^{π} as the tabular transition probabilities under the policy π.

Proof of Theorem 1

In Eqn. 3.5, The Q-function update can be computed in a tabular setting, by setting the derivative of the augmented objective in Eqn. 3.5 with respect to Q to zero,

$$\varepsilon\left(d^{\phi}\pi - d^{\pi_\beta}\pi\right) + \alpha\left(d^{\pi_\beta}\pi - d^{\pi_\beta}\pi_\beta\right) + d^{\pi_\beta}\pi_\beta\left(Q - \mathcal{B}\hat{Q}^k\right) = 0$$

Therefore, we can obtain $\hat{Q}^{k+1}$ in terms of $\hat{Q}^k$ by rearranging the terms,

$$\hat{Q}^{k+1}(\mathbf{s},\mathbf{a}) = \hat{\mathcal{B}}^\pi \hat{Q}^k(\mathbf{s},\mathbf{a}) - \varepsilon \frac{\left(d^\phi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s})\right)\pi(\mathbf{a}|\mathbf{s})}{d^{\pi_\beta}(\mathbf{s})\pi_\beta(\mathbf{a}|\mathbf{s})} - \alpha\left[\frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right], \tag{3.6}$$

for all $\mathbf{s} \in \mathcal{S}$, $\mathbf{a} \in \mathcal{A}$ and $k \in [0, +\infty)$. For the state-action pair $(\mathbf{s},\mathbf{a})$ such that $d^\phi(\mathbf{s}) < d^{\pi_\beta}(\mathbf{s})$ and $\pi(\mathbf{a}|\mathbf{s}) < \pi_\beta(\mathbf{a}|\mathbf{s})$, the last two terms of Eqn. 3.6, $-\varepsilon \frac{\left(d^\phi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s})\right)\pi(\mathbf{a}|\mathbf{s})}{d^{\pi_\beta}(\mathbf{s})\pi_\beta(\mathbf{a}|\mathbf{s})} - \alpha\left[\frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right]$ is positive, so that we cannot simply lower bound the true Q-function $Q^{k+1}(\mathbf{s},\mathbf{a})$ by the estimated one $\hat{Q}^{k+1}(\mathbf{s},\mathbf{a})$ point-wise. However, we can prove that the value function, which is the expectation of the Q-function, can be lower bounded. Taking the expectation of both sides of Eqn. 3.6 under the distribution $\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})$, we have

$$\hat{V}^{k+1}(\mathbf{s}) = \mathcal{B}^\pi \hat{V}^k(\mathbf{s}) - \varepsilon \mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[\frac{\left(d^\phi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s})\right)\pi(\mathbf{a}|\mathbf{s})}{d^{\pi_\beta}(\mathbf{s})\pi_\beta(\mathbf{a}|\mathbf{s})}\right] - \alpha \mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[\frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right]. \tag{3.7}$$

The first goal is to prove that $\hat{V}^{k+1} \leq \mathcal{B}^\pi \hat{V}^k$ which implies that each iteration introduces some underestimation, and $\hat{V}^k$ could eventually converge to a fixed point. Therefore, we need to prove the last two terms on the right hand side of Eqn. 3.7 is negative. We denote Δ to be the opposite of the last two terms on the right hand side of Eqn. 3.7, then

$$\begin{aligned}\Delta &= \varepsilon \mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[\frac{\left(d^\phi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s})\right)\pi(\mathbf{a}|\mathbf{s})}{d^{\pi_\beta}(\mathbf{s})\pi_\beta(\mathbf{a}|\mathbf{s})}\right] + \alpha \mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[\frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right] \\ &= \varepsilon \frac{d^\phi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s})}{d^{\pi_\beta}(\mathbf{s})} \sum_{\mathbf{a}} \frac{\pi^2(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} + \alpha \sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s})\left[\frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right].\end{aligned} \tag{3.8}$$

From the proof in [74], we know that the second term in Eqn. 3.8 is non-negative when $\alpha > 0$, that is

$$D_{\text{CQL}}(\mathbf{s}) = \sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s})\left[\frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right] \geq 0. \tag{3.9}$$

Hence, when $d^\phi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s}) \geq 0$, it is obvious that $\Delta \geq 0$ for all $\varepsilon > 0$. When $d^\phi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s}) < 0$, if ε satisfies

$$\varepsilon \leq \alpha D_{\text{CQL}}(\mathbf{s}) \left(\frac{\left|d^\phi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s})\right|}{d^{\pi_\beta}(\mathbf{s})} \sum_{\mathbf{a}} \frac{\pi^2(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})}\right)^{-1}, \tag{3.10}$$

then we have $\Delta \geq 0$.

In summary, we have $\Delta \geq 0$ when Eqn. 3.10 holds. Since the exact Bellman update operator $\mathcal{B}^\pi$ is a contraction [129], we have

$$||\mathcal{B}^\pi \hat{V}^{k+1} - \mathcal{B}^\pi \hat{V}^k|| = ||(\mathcal{B}^\pi \hat{V}^{k+1} - \Delta) - (\mathcal{B}^\pi \hat{V}^k - \Delta)|| \leq \gamma ||\hat{V}^{k+1} - \hat{V}^k||$$

which implies that each value-function update $\hat{V}^{k+1} = \mathcal{B}^\pi \hat{V}^k - \Delta$ is a contraction. According to the contraction mapping theorem, the recursive update in Eqn. 3.7 will always lead value function to converge to a fixed point $\hat{V}^\pi$. Now that $V^\pi = \mathcal{B}^\pi V^\pi$ for the true value functions, by subtracting them from both side of Eqn. 3.7 and substitute $\hat{V}^{k+1}$ and $\hat{V}^k$ with the fixed point $\hat{V}^\pi$, we have

$$\hat{V}^\pi = V^\pi - \underbrace{(I - \gamma P^\pi)^{-1}}_{\text{Positive semi-definite}} \left[\underbrace{\alpha \mathbb{E}_\pi \left[\frac{\pi}{\pi_\beta} - \mathbf{1} \right] + \varepsilon \mathbb{E}_\pi \left[\frac{(d^\phi - d^{\pi_\beta})\pi}{d^{\pi_\beta} \pi_\beta} \right]}_{\geq \mathbf{0}} \right], \tag{3.11}$$

when ε satisfies

$$\frac{\varepsilon}{\alpha} \leq \min_{\mathbf{s}} \left(\sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) \left[\frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1 \right] \right) \left(\frac{\left| d^\phi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s}) \right|}{d^{\pi_\beta}(\mathbf{s})} \sum_{\mathbf{a}} \frac{\pi^2(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} \right)^{-1}. \tag{3.12}$$

In Eqn. 3.11, we stretch all notations to be vectors, which means $\hat{V}^\pi \in \Re^{|\mathcal{S}|}$, $V^\pi \in \Re^{|\mathcal{S}|}$, and $\alpha \mathbb{E}_\pi \left[\frac{\pi}{\pi_\beta} - \mathbf{1} \right] + \varepsilon \mathbb{E}_\pi \left[\frac{(d^\phi - d^{\pi_\beta})\pi}{d^{\pi_\beta} \pi_\beta} \right] \in \Re^{|\mathcal{S}|}$ are all vectors containing values for all states. Here, $\mathbf{1}$ denotes the vector in which the entries are all 1. The expectations and the operations inside $\alpha \mathbb{E}_\pi \left[\frac{\pi}{\pi_\beta} - \mathbf{1} \right] + \varepsilon \mathbb{E}_\pi \left[\frac{(d^\phi - d^{\pi_\beta})\pi}{d^{\pi_\beta} \pi_\beta} \right]$ are all computed in a point-wise manner.

Therefore, we can conclude from Eqn. 3.11 that the estimated value function $\hat{V}^\pi(\mathbf{s})$ is a lower bound of the true value-function $V^\pi(\mathbf{s})$ without considering any sampling error. Thus, we finish the proof of Thm. 1.

Value Lower Bound in Existence of Sampling Errors

We now take sampling error into account. First, we introduce a lemma from [74]:

Lemma 3.4.1 *If with high probability δ the reward function $r(\mathbf{s}, \mathbf{a})$ and the transition function $T(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ can be estimated with bounded error, then the sampling error of the empirical Bellman operator is also bounded:*

$$\left| \hat{\mathcal{B}}^\pi V(\mathbf{s}) - \mathcal{B}^\pi V(\mathbf{s}) \right| \leq \frac{C_{r,T,\delta} R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}(\mathbf{s}, \mathbf{a})|}}, \tag{3.13}$$

where $C_{r,T,\delta}$ is a constant related to r, T, and δ, and $R_{\max}$ is the maximum possible reward in the environment.

Note that the bound of the error $\left|\hat{\mathcal{B}}^\pi V(\mathbf{s}) - \mathcal{B}^\pi V(\mathbf{s})\right|$ in Lemma 3.4.1 only holds for states and actions in the training datasets, i.e. $\mathbf{s}, \mathbf{a} \in \mathcal{D}$. We have no reward or transition pair collected at unseen states or actions outside the training dataset, so it is impossible to bound the error outside the training dataset when consider the sampling error introduced by the reward function and the transition function. Therefore, we can lower bound the true value function by the learned value function at states and actions in the training datasets as in the following corollary.

Corollary 3.4.1 *When the sampling error defined in Lemma 3.4.1, for any state and any action in the training dataset,* $\mathbf{s}, \mathbf{a} \in \mathcal{D}$*, the learned value function via Eqn. 3.5 is a lower bound of the true one, i.e.,* $\hat{V}^\pi(\mathbf{s}) \le V^\pi(\mathbf{s})$*, if the trade-off factor* ε *and* α *satisfy the constraints*

$$\begin{aligned}
&\varepsilon \le \alpha \min_{\mathbf{s}\in\mathcal{D}} \left(\sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) \left[\frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right]\right) \left(\frac{\left|d^\phi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s})\right|}{d^{\pi_\beta}(\mathbf{s})} \sum_{\mathbf{a}} \frac{\pi^2(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})}\right)^{-1} \\
&\alpha \ge \max_{\mathbf{s},\mathbf{a}\in\mathcal{D}} \frac{C_{r,T,\delta} R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}(\mathbf{s},\mathbf{a})|}} \cdot \max_{\mathbf{s},\mathbf{a}\in\mathcal{D}} \left(\sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) \left[\frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right]\right)^{-1}.
\end{aligned} \tag{3.14}$$

We now show the proof of Corollary 3.4.1. From Eqn. 3.11, we can directly bound the estimated value function for any $\mathbf{s}, \mathbf{a} \in \mathcal{D}$ by

$$\hat{V}^\pi = V^\pi - (I - \gamma P^\pi)^{-1} \left[\alpha \mathbb{E}_\pi \left[\frac{\pi}{\pi_\beta} - \mathbf{1}\right] + \varepsilon \mathbb{E}_\pi \left[\frac{(d^\phi - d^{\pi_\beta})\pi}{d^{\pi_\beta}\pi_\beta}\right] - \frac{C_{r,T,\delta} R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}(\mathbf{s},\mathbf{a})|}}\right], \tag{3.15}$$

when ε and α satisfy the constraints in Eqn. 3.14. Note that in Eqn. 3.15, we use vector notations similar to those in Eqn. 3.11.

Therefore, when we consider sampling error introduced by the reward function and the transition pair, the learned value function by PessORL still lower bounds the true one for any states and actions in the training dataset. Thus, we finish the proof of Corollary 3.4.1.

Proof of Theorem 2

We begin the proof from Eqn. 3.7. We first take the expectation of both side of Eqn. 3.7 under the distribution d^ϕ, then

$$\begin{aligned}
\mathbb{E}_{d^\phi(\mathbf{s})} \left[\hat{V}^{k+1}(\mathbf{s})\right] &= \mathbb{E}_{d^\phi(\mathbf{s})} \left[\mathcal{B}^\pi \hat{V}^k(\mathbf{s})\right] \\
&- \varepsilon \sum_{\mathbf{s}} d^\phi(\mathbf{s}) \frac{d^\phi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s})}{d^{\pi_\beta}(\mathbf{s})} \sum_{\mathbf{a}} \frac{\pi^2(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - \alpha \sum_{\mathbf{s}} d^\phi(\mathbf{s}) \sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) \left[\frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right].
\end{aligned} \tag{3.16}$$

Similarly, we take the expectation of both side of Eqn. 3.7 under the distribution d^{π_β}, then we have

$$\begin{aligned}\mathbb{E}_{d^{\pi_\beta}(\mathbf{s})}\left[\hat{V}^{k+1}(\mathbf{s})\right] &= \mathbb{E}_{d^{\pi_\beta}(\mathbf{s})}\left[\mathcal{B}^\pi \hat{V}^k(\mathbf{s})\right] \\ &- \varepsilon \sum_{\mathbf{s}} d^{\pi_\beta}(\mathbf{s}) \frac{d^\phi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s})}{d^{\pi_\beta}(\mathbf{s})} \sum_{\mathbf{a}} \frac{\pi^2(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - \alpha \sum_{\mathbf{s}} d^{\pi_\beta}(\mathbf{s}) \sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) \left[\frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right]. \end{aligned} \tag{3.17}$$

If we subtract Eqn. 3.17 from Eqn. 3.16, we get

$$\begin{aligned}\mathbb{E}_{d^\phi}\left[\hat{V}^{k+1}(\mathbf{s})\right] - \mathbb{E}_{d^{\pi_\beta}}\left[\hat{V}^{k+1}(\mathbf{s})\right] &= d^{\phi\top}\mathcal{B}^\pi \hat{V}^k(\mathbf{s}) - d^{\pi_\beta\top}\mathcal{B}^\pi \hat{V}^k(\mathbf{s}) \\ &- \varepsilon \sum_{\mathbf{s}} \frac{(d^\pi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s}))^2}{d^{\pi_\beta}(\mathbf{s})} \sum_{\mathbf{a}} \frac{\pi^2(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - \alpha \sum_{\mathbf{s}} (d^\phi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s})) \sum_{\mathbf{a}} \frac{\left(\pi(\mathbf{a}|\mathbf{s}) - \pi_\beta(\mathbf{a}|\mathbf{s})\right)^2}{\pi_\beta(\mathbf{a}|\mathbf{s})}\end{aligned} \tag{3.18}$$

Therefore, we have $\mathbb{E}_{d^\phi}\left[\hat{V}^{k+1}(\mathbf{s})\right] \le \mathbb{E}_{d^{\pi_\beta}}\left[\hat{V}^{k+1}(\mathbf{s})\right]$, if ε satisfies

$$\begin{aligned}\varepsilon \ge &\left[d^{\phi\top}\mathcal{B}^\pi \hat{V}^k(\mathbf{s}) - d^{\pi_\beta\top}\mathcal{B}^\pi \hat{V}^k(\mathbf{s}) - \alpha \sum_{\mathbf{s}} (d^\phi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s})) \sum_{\mathbf{a}} \frac{\left(\pi(\mathbf{a}|\mathbf{s}) - \pi_\beta(\mathbf{a}|\mathbf{s})\right)^2}{\pi_\beta(\mathbf{a}|\mathbf{s})}\right] \\ &\times \left[\sum_{\mathbf{s}} \frac{(d^\pi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s}))^2}{d^{\pi_\beta}(\mathbf{s})} \sum_{\mathbf{a}} \frac{\pi^2(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})}\right]^{-1}. \end{aligned} \tag{3.19}$$

Existence of Feasible Trade-off Factor

Note that both Eqn. 3.19 and Eqn. 3.14 put constraints on the trade-off factor ε. We show that we can choose an appropriate value of α to ensure that a feasible ε that satisfies both constraints exist. Formally, we denote

$$\begin{aligned} X &= d^{\phi\top}\mathcal{B}^\pi \hat{V}^k(\mathbf{s}) - d^{\pi_\beta\top}\mathcal{B}^\pi \hat{V}^k(\mathbf{s}), \\ Y &= \sum_{\mathbf{s}} (d^\phi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s})) \sum_{\mathbf{a}} \frac{\left(\pi(\mathbf{a}|\mathbf{s}) - \pi_\beta(\mathbf{a}|\mathbf{s})\right)^2}{\pi_\beta(\mathbf{a}|\mathbf{s})}, \\ Z &= \sum_{\mathbf{s}} \frac{(d^\pi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s}))^2}{d^{\pi_\beta}(\mathbf{s})} \sum_{\mathbf{a}} \frac{\pi^2(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})}, \\ W &= \min_{\mathbf{s}\in\mathcal{D}} \left(\sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) \left[\frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right]\right) \left(\frac{\left|d^\phi(\mathbf{s}) - d^{\pi_\beta}(\mathbf{s})\right|}{d^{\pi_\beta}(\mathbf{s})} \sum_{\mathbf{a}} \frac{\pi^2(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})}\right)^{-1} \\ U &= \max_{\mathbf{s},\mathbf{a}\in\mathcal{D}} \frac{C_{r,T,\delta} R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}(\mathbf{s},\mathbf{a})|}} \cdot \max_{\mathbf{s},\mathbf{a}\in\mathcal{D}} \left(\sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) \left[\frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - 1\right]\right)^{-1} \end{aligned} \tag{3.20}$$

for simplicity. From Eqn. 3.14 and 3.19, we have

$$(X - \alpha Y)Z^{-1} \leq \varepsilon \leq \alpha W$$
$$\alpha \geq U$$

Hence, there exists a feasible ε when $(X - \alpha Y)Z^{-1} \leq \alpha W$ and $\alpha \geq U$. Thus, when

$$\alpha \geq \min\left(X(WZ+Y)^{-1}, U\right), \tag{3.21}$$

we can choose a feasible ε from the interval $[(X - \alpha Y)\, Z^{-1}, \alpha W]$.

3.5 Implementing the Algorithm

In this section, we introduce a practical PessORL algorithm based on Eqn. 3.5. This algorithm simply modifies the policy evaluation step of Deep Q-Learning or Soft Actor-Critic algorithms, which is easy to implement.

Detecting OOD states

In prior to designing the algorithm, we need to choose a proper $d^{\phi}(\mathbf{s})$, which requires a tool for OOD state detection. Following [68, 78, 28], we use bootstrapping to detect OOD states. In particular, we train a bag of Gaussian dynamics models [28] $\{\hat{P}_1, \hat{P}_2, \ldots, \hat{P}_n\}$ where each model is $\hat{P}_i(\cdot|\mathbf{s}, \mathbf{a}) = \mathcal{N}(\mathbf{s} + \hat{f}_{\phi_i}(\mathbf{s}, \mathbf{a}), \hat{\Sigma}_{\phi_i})$. The function $\hat{f}_{\phi_i}$ outputs the mean difference between the next state and the current state, and Σ_{ϕ_i} models the standard deviation. OOD states are detected by estimating the uncertainty of bootstrap models at a given state $\mathbf{s} \in \mathcal{S}$. Concretely, we define $u_{\pi}(\mathbf{s}) = \mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[\frac{1}{n}\sum_{i=1}^{n}\left(\hat{f}_{\phi_i}(\mathbf{s}, \mathbf{a}) - \bar{f}_{\phi}(\mathbf{s}, \mathbf{a})\right)^2\right]$, where $\bar{f}_{\phi}(\mathbf{s}, \mathbf{a}) = \dfrac{1}{n}\sum_{i=1}^{n}\hat{f}_{\phi_i}(\mathbf{s}, \mathbf{a})$ is the mean of outputs of all $\hat{f}_{\phi_i}$, and the actions are drawn from a policy distribution π. A high $u_{\pi}(\mathbf{s})$ value indicates the state is more likely to be an unseen state. Given a set of sampled states $\{s_1, s_2, \ldots, s_n\}$, we can define a discrete distribution over it using $u_{\pi}(\mathbf{s})$: $\zeta(\mathbf{s}_i) = \dfrac{u(\mathbf{s}_i)}{\sum_j u(\mathbf{s}_j)}, i = 1, 2, ..., n$, which assign high probabilities to OOD states. In the following section, we will use it to construct the distribution $d^{\phi}(\mathbf{s})$.

Practical Implementation of PessORL

We now introduce a practical PessORL algorithm. In practice, to obtain a well-defined distribution d^{ϕ}, we add an additional optimization problem over d^{ϕ} into the original optimization problem. The resulting optimization problem for the policy evaluation step is:

$$\min_{Q}\max_{d^{\phi}}\left[\varepsilon\left(\mathbb{E}_{\mathbf{s}\sim d^{\phi}(\mathbf{s}),\mathbf{a}\sim\hat{\pi}^k(\mathbf{a}|\mathbf{s})}\left[Q(\mathbf{s}, \mathbf{a})\right] - \mathbb{E}_{\mathbf{s}\sim d^{\pi_\beta}(\mathbf{s}),\mathbf{a}\sim\hat{\pi}^k(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s}, \mathbf{a})]\right) + \mathcal{R}(d^{\phi})\right]$$
$$+ \mathcal{E}(Q, \hat{\mathcal{B}}^{\pi}\hat{Q}^k_{\theta}) + \mathcal{C}(Q), \tag{3.22}$$

where $\mathcal{R}(d^{\phi})$ is a regularization term inspired by [74] in order to stabilize the training. If we choose $\mathcal{R}(d^{\phi}) = -D_{\mathrm{KL}}(d^{\phi}(\mathbf{s}) \,||\, \zeta(\mathbf{s}))$, where $\zeta(\mathbf{s})$ is the distribution we obtained from uncertainty estimations, then $d^{\phi}(\mathbf{s}) \propto \zeta(\mathbf{s}) \exp\left(V^{\hat{\pi}^k}(\mathbf{s})\right)$, where $V^{\hat{\pi}^k}(\mathbf{s}) = \mathbb{E}_{\mathbf{a}\sim\hat{\pi}^k(\mathbf{a}|\mathbf{s})}\left[Q(\mathbf{s}, \mathbf{a})\right]$. The resulting d^{ϕ} is intuitively reasonable, because it assigns high probabilities to OOD states with high uncertainty estimations. In particular, d^{ϕ} assigns higher probabilities to states with high values, because we expect to penalize harder on them than those with low values already. With this choice of d^{ϕ} in Eqn. 3.22, we obtain the following PessORL policy evaluation step:

$$\min_{Q} J(Q) = \min_{Q} \varepsilon \left(\log \sum_{\mathbf{s}} \zeta(\mathbf{s}) \exp\left(V^{\hat{\pi}^k}(\mathbf{s})\right) - \mathbb{E}_{\mathbf{s}\sim d^{\pi_\beta}(\mathbf{s})}[V^{\hat{\pi}^k}(\mathbf{s})] \right) + \mathcal{E}(Q, \hat{\mathcal{B}}^{\pi}\hat{Q}^k_{\theta}) + \mathcal{C}(Q). \quad (3.23)$$

The first term in Eqn. 3.23 is very similar to weighted softmax values over the state space. It penalizes the softmax value over the state space, but also considers the distances between sample points and the training data. The two terms following the trade-off factor ε is actually trying to decrease the discrepancy between the softmax value over OOD states and the average value over in-distribution states. Intuitively, it should enforce the learned value function to output higher values at in-distribution states, and lower values at out-of-distribution states. The logsumexp term in Eqn. 3.23 also mitigates the requirement for an accurate uncertainty estimation $\zeta(\mathbf{s})$ over the entire state space. Only those states with high values contribute to the regularization.

The complete algorithm is shown in Algorithm 1. We include the version for continuous action space which requires a policy network here, and note that if the action space is discrete, then we no longer need a policy network but just an implicit policy based on the learned Q-function. We implement PessORL on top of CQL [74], with its default hyperparameters. We also apply Lagrangian dual gradient descent to automatically adjust the trade-off factor ε. During the training process of offline reinforcement learning algorithms such as CQL and PessORL, we only have access to the dataset $\mathcal{D}$ instead of $d^{\pi_\beta}(\mathbf{s})$ and $\pi_\beta(\mathbf{a}|\mathbf{s})$. Therefore, we follow the convention in reinforcement learning community and approximated all expectations by Monte Carlo estimation in Eqn. 3.23.

Implementation Details of the Algorithm

The implementation of our algorithm is based on the original implementation of CQL: `https://github.com/aviralkumar2907/CQL`. We first train a bag of dynamics models $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n\}$, and then train the Q-network and policy network. When we train the Q-network and policy network, the uncertainty estimation model $u_\pi(\mathbf{s})$ is induced by the pre-trained dynamics models. We found in the experiments that a fixed trade-off factor ε would cause the learned value function to be too conservative and hence the learned policy to fail, so we choose to use a varying ε which is adjusted by dual gradient-descent. Following the implementation of CQL, we introduce a "budget" parameter τ to automatically control

ε as below:

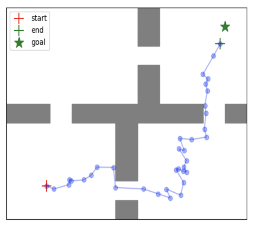
$$\min_{Q} \max_{\varepsilon \geq 0} \left[\varepsilon \left(\log \sum_{\mathbf{s}} \zeta(\mathbf{s}) \exp \left(V^{\hat{\pi}^k}(\mathbf{s}) \right) - \mathbb{E}_{\mathbf{s} \sim d^{\pi_\beta}(\mathbf{s})}[V^{\hat{\pi}^k}(\mathbf{s})] \right) - \tau \right] + \mathcal{E}(Q, \hat{\mathcal{B}}^{\pi} \hat{Q}^k_{\theta}) + \mathcal{C}(Q). \quad (3.24)$$

From Eqn. 3.24, we can see that if the discrepancy between values is less than τ, then ε will be set to zero. When the discrepancy is larger than the threshold τ, ε will be increased to a large value to penalize harder on the value function. This automatic mechanism ensures a reasonable choice of ε, and reduce the tedious procedure to tune the hyperparameter ε. In the Gym MuJoCo domain, we choose $\tau = 0.0$ for the Hopper environment, and $\tau = 10.0$ for the Walker2d, Halfcheetah and Ant environment. In the Adroit domain, we choose $\tau = 20.0$ for all four environments.

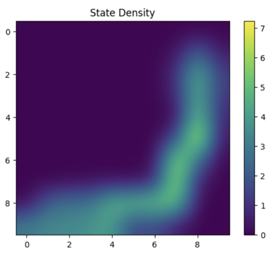
For the dynamics models learning part, we follow the convention in model-based reinforcement learning domain, and adopt a four layers MLP with a size of 400 each. We choose to train five bootstrap models and collect them in a bag to estimate uncertainty later in the policy evaluation and improvement steps. Each dynamic model is a Gaussian model which outputs the mean and the logstd of the next state deviation. When we train the models, we iterate through all training data for 10 epochs with a learning rate of $1e-4$ and a batch size of 256.

For the Q-function learning part of the algorithm, we inherit the twin Q-function trick and soft target updates from the original SAC implementation on D4RL tasks. The Q-functions are optimized by Adam with a batch size of 256, and the learning rate is chosen to be $3e-4$ across the environments. We design the Q-functions to have three layers with a size of 256 each. When we evaluate the logsumexp term in Eqn. 3.24, we need to sample states from the state space. However, the state space is unbounded on the gym domain and Adroit domain, so we set the sample range to be $[\mu - 10 * \sigma, \mu + 10 * \sigma]$, where μ and σ are the mean and the standard deviation of the current batch sampled from the training dataset.

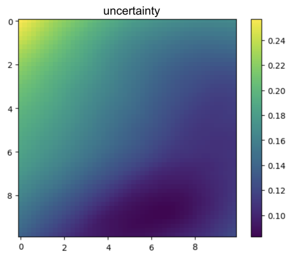
For the policy learning part of the algorithm, we do not need an explicit policy network to model the policy, but just an implicit argmax policy in discrete settings such as Pointmass environment. In continuous settings on the Gym and the Adroit domain, we adopt a Tanh-Gaussian policy structure by the rlkit repository: `https://github.com/vitchyr/rlkit`. Since the action spaces in these domains are all bounded by -1 and 1, the Tanh-Gaussian policy can capture this constraint naturally. The policy network is designed to have three layers with a size of 256 each. The policy network is also optimized by Adam with a batch size of 256, with a learning rate of $3e-4$ for the Pointmass and the Gym domain, and $3e-5$ for the Adroit domain.



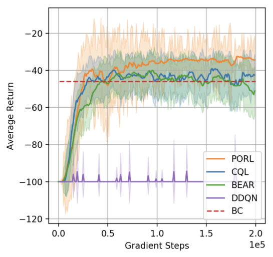
(a-1)



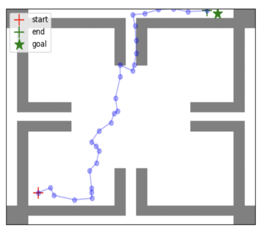
(b-1)



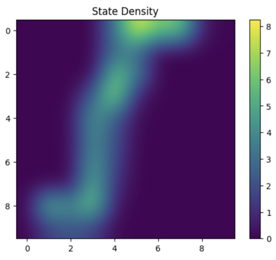
(c-1)



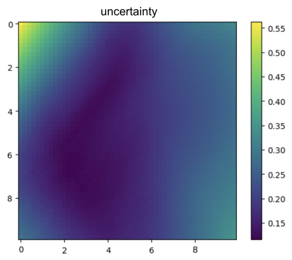
(d-1)



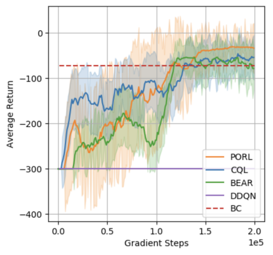
(a-2)



(b-2)



(c-2)



(d-2)

Figure 3.1: (a) The whole map of the environment; (b) The state density in training dataset; (c) The visualization of uncertainty estimation; (d) The learning curves. The top row (1) and the bottom row (2) are corresponding to PointmassHard-v0 and PointmassSuperHard-v0, respectively. We can see that almost all trajectories in the training datasets are located around the optimal trajectory from the start to the goal in the yellow areas in (b), indicating they are collected by a near-optimal policy.

3.6 Experiments

We compare our algorithm to prior offline algorithms: two state-of-the-art offline RL algorithms BEAR [71] and CQL [74]; two baselines adapted directly from online algorithms, actor-critic algorithm TD3 [45] and DDQN [141]; and behavior cloning (BC). The TD3 baseline is applied when the action space is continuous, whereas DDQN is trained when the action space is discrete. We evaluate each algorithm on a wide range of task domains, including tasks with both continuous and discrete state and action space. All baselines are run with the default code and hyperparameters from the original repositories. In particular, we are interested in the comparison between our algorithm with CQL, because we essentially add an additional state regularization term to the original CQL framework.

Performance on Various Environments

Pointmass Mazes. The task for the agent in this domain is to learn from expert demonstrations to navigate from a random start to a fixed goal. The expert dataset, which contains around 1000 trajectories all from the same start point to the same goal, is collected by online trained RL policy. During the test time of offline RL algorithms, we reset the start to a random point in the state space and the goal to the same fixed point as the dataset. In this way, the performance of the agent at unseen states are evaluated.

Before showing the performance, we first check if the OOD states detection is accurate, and hence, if we can successfully penalize high values at unseen states in training datasets. We evaluate the effectiveness of the OOD states detection method based on the accuracy of uncertainty estimation in the environment Pointmass. Figure 3.1(b) and (c) are visualizations of the training datasets and estimated uncertainty $u_\pi(\mathbf{s})$, both of which have the coordinate systems the same as that in the map (figure 3.1(a)). We use different colors in figure 3.1(b) and (c) to represent different values at each point in the map. The uncertainty estimations tend to be high (yellow areas in Fig. 3.1(c)) in area where the state density is low (blue areas in Fig 3.1(b)), and vice versa. This trend empirically shows that our uncertainty estimations are reasonable. We can trust them to detect OOD states when training offline RL algorithms.

We include the learning curves in figure 3.1(d), in which we evaluate each algorithm based on 3 random seeds, and report the average return. The shaded area represents the standard deviation of each evaluation. As we can see in the figures, PessORL outperforms other baselines in both hard and super hard environments. PessORL benefits from the augmented policy evaluation step in Eqn. 3.23. The learned value function produces high values at areas that have low uncertainty estimations, and low values at highly uncertain areas (OOD states). Therefore, the agent can be "attracted" to the high value areas from low value and unfamiliar areas.

Gym Tasks. In this domain, we focus on the locomotion environments from MuJoCo, including Walker2d-v2, Hopper-v2, Halfcheetah-v2, and Ant-v2. Unlike Pointmass environment, we directly adopt the d4rl datasets [44] as our training data in the gym domains. We include four different types of datasets in our experiments, namely, "medium", "medium-expert", "random", and "expert". The "medium", "random", and "expert" dataset are all collected by a single policy, which is an either early-stopping trained, or randomly initialized, or fully trained expert policy. The "medium-expert" dataset is generated by mixing mediocre and expert quality data. We show the normalized scores averaged over 4 random seeds for all methods on gym domain in table 3.1. We directly ran all baselines from their original repositories with their default parameters, and we only report the average scores we actually obtained. As we can see in the table, PessORL outperforms all other offline RL methods on a majority of tasks on gym domains. PessORL works especially well with mediocre quality datasets according to the results. In fact, it is one of the advantages of offline RL methods over behavior cloning on medium quality datasets, because offline RL methods take advantage of the information both from the reward and the underlying state and action distributions in training datasets, instead of simply imitating behavior policies as behavior cloning. Medium quality datasets are also considered to be similar to real-world datasets. Therefore, it is important for an offline RL method to perform well in medium quality datasets. We also note that PessORL shares some good properties with CQL, such as satisfying performance on mixed quality datasets. PessORL and CQL both outperform other offline methods on medium-expert datasets with PessORL better between them. The

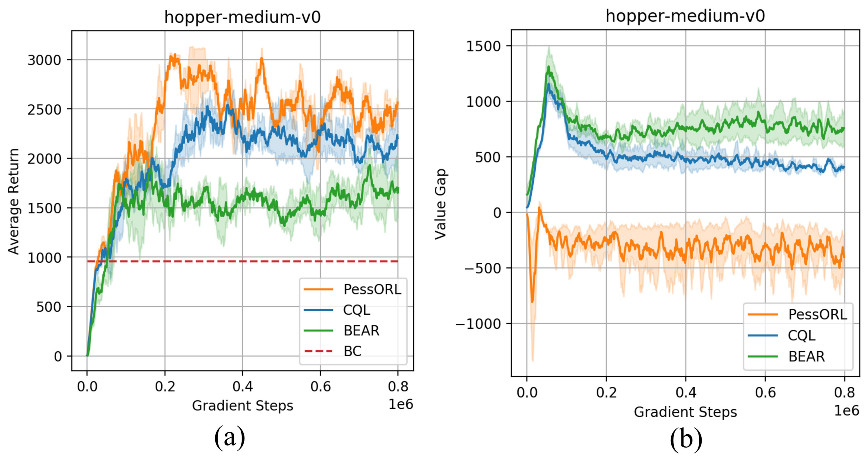


Figure 3.2: (a) The learning curves in hopper-medium-v0. (b) The discrepancy Δ_k as a function of gradient steps for PessORL, CQL, and BEAR.

reason is that offline RL methods can "stitch" [44] different trajectories from different policies together according to the information from the reward.

Adroit Tasks The adroit domain [114] provides more challenging tasks than the Point-mass environment and the gym domain. The tasks include controlling a 24-DoF simulated Shadow Hand robot to twirl a pen, open a door, hammer a nail, and relocate a ball. Similar to the datasets in the gym domain, we also directly use the d4rl datasets as the training datasets in our experiments. The performance of PessORL and all baselines is shown in table 3.1. The normalized scores of all methods are average returns on 5 random seeds. We note that PessORL has better performance than other baselines on adroit domains. It is a great advantage for PessORL to learn useful skills from human demonstrations on these high dimensional and highly realistic robotic simulations.

Discussions and Limitations

The main contribution of this work is to explicitly limit the values at OOD states, so that the learned policy can act conservatively at OOD states and drives the agent back to the familiar areas near the training data. We are interested to see if our framework can indeed induce a different behavior on OOD states. We use $\Delta_k = \max_{\mathbf{s}\in\mathcal{S}}[V(\mathbf{s})] - \mathbb{E}_{\mathbf{s}\sim\mathcal{D}}[V(\mathbf{s})]$ as a metric to evaluate it at each iteration. If Δ_k is close to zero, then intuitively it indicates the values at OOD states are lower than those at in-distribution states. In Fig. 3.2, we plot Δ_k at each iteration in hopper-medium-v0. As is shown in the figure, PessORL successfully limits Δ_k to be non-positive, which meets our goal in this work and aligns with the statement in theorem 2.

On the gym domain, we notice that the performance of PessORL and CQL on datasets containing expert trajectories is not satisfying, often not as good as BC. We believe it is because of overly conservative value estimation. In fact, it is widely believed that conservative methods suffer from underestimation [78]. The conservative objective function in Eqn. 3.23 sometimes assign values that are too low to OOD states and actions. Besides, the uncertainty estimation method cannot be guaranteed to be precise on high-dimensional spaces. It is actually a possible future work direction to solve the underestimation and uncertainty estimation problems in conservative methods.

Ablation Studies

What if $d^\phi(\mathbf{s})$ is induced by a uniform distribution?

In Sec. 3.5, we introduced a practical method to construct $d^\phi(\mathbf{s})$. Alternatively, we may simply set $\zeta(\mathbf{s})$ as a uniform distribution, which also satisfies our requirement and leads to $d^\phi(\mathbf{s}) \propto \exp\left(V^{\hat{\pi}^k}(\mathbf{s})\right)$. A uniform distribution could be more convenient if it is time consuming to construct $d^\phi(\mathbf{s})$ from data. Formally, if we choose $d^\phi(\mathbf{s}) \propto \exp\left(V^{\hat{\pi}^k}(\mathbf{s})\right)$ which is induced by a uniform distribution, then the practical PessORL policy evaluation step becomes:

$$\min_{Q} \max_{\varepsilon \geq 0} \left[\varepsilon \left(\log \sum_{\mathbf{s}} \exp\left(V^{\hat{\pi}^k}(\mathbf{s})\right) - \mathbb{E}_{\mathbf{s}\sim d^{\pi_\beta}(\mathbf{s})}[V^{\hat{\pi}^k}(\mathbf{s})]\right) - \tau\right] \;+\; \mathcal{E}(Q, \hat{\mathcal{B}}^\pi \hat{Q}_\theta^k) \;+\; \mathcal{C}(Q). \tag{3.25}$$

We name this variant of PessORL as PessORL-uniform. We evaluate the variant PessORL-uniform on two D4RL MuJoCo environments, and compare its performance to the original PessORL algorithm, CQL, BEAR, and BC based on the average returns over three random seeds. In Fig. 3.3, we show the learning curves and the value gap Δ_k in hopper-medium-v0 and walker2d-medium-v0. The value gap $\Delta_k = \max_{\mathbf{s}\in\mathcal{S}}[V(\mathbf{s})] - \mathbb{E}_{\mathbf{s}\sim\mathcal{D}}[V(\mathbf{s})]$ follows the same definition in Sec. 3.6, which evaluates whether a method can assign high values at the states in the training data and low values at the out-of-distribution states.

In Fig. 3.3(a) and (c), we can see that the average return of PessORL-uniform is lower than those of CQL and PessORL. Fig. 3.3(b) and (d) show that the value function learned by PessORL-uniform is either not pessimistic enough or too pessimistic, causing the average return to be lower than the other two rivals. We believe the reason is that a uniform distribution in PessORL-uniform does not discriminate between OOD states and in-distribution states, so the objective function in Eqn. 3.25 works differently from the original one in PessORL. Eqn. 3.25 increases the values at in-distribution states, instead of penalizing the values at OOD states as before. This mechanism cannot guarantee an accurate penalization on most OOD states, so we observe different gaps Δ_k in different environments.

In conclusion, constructing $d^\phi(\mathbf{s})$ via a uniform distribution leads to worse performance, but it could be a cheap alternative in terms of computational cost.

What if an uncertainty term is directly subtracted from the learned Q-function?

In Sec. 3.1 and Sec. 3.3, we discussed two main categories of method to obtain pessimistic value functions. Besides adding a regularization term in policy evaluation step as in the proposed PessORL, we can also improve the policy by a pessimistic estimate of Q-function. This pessimistic Q-function can be obtained by directly subtracting an uncertainty term from the learned value function, i.e., $\hat{Q}^c_\theta(\mathbf{s},\mathbf{a}) = \hat{Q}^k_\theta(\mathbf{s},\mathbf{a}) - \beta \text{Unc}(\mathbf{s},\mathbf{a})$, where we use the superscript c to denote a conservative Q-function, and β is a trade-off factor that makes the scale of the Q-function and the uncertainty term comparable. In this section, we first introduce two variants of the proposed PessORL algorithm, named PessORL-unc and PessORL-OPIQ, in which we used the aforementioned pessimistic Q-function to improve the policy. Then we compare the variants PessORL-unc and PessORL-OPIQ to the original PessORL, and discuss why we choose to obtain a pessimistic Q-function via adding a regularization term in policy evaluation step instead of directly subtracting an uncertainty term from the original learned value function.

We first introduce a variant of our algorithm, named PessORL-unc, in which the additional regularization term in the original PessORL policy evaluation step is removed. The policy evaluation step in PessORL-unc then becomes as follows:

$$\min_Q \mathcal{E}(Q, \hat{B}^\pi \hat{Q}^k_\theta) + \mathcal{C}(Q). \tag{3.26}$$

Actually, Eqn. 3.26 is the same as the policy evaluation step in CQL. In the ablation studies, we use this optimization problem to learn a Q-function first, and then we directly subtract an uncertainty term from the learned Q-function to get the final Conservative Q-function to be used in the policy improvement step:

$$\hat{\pi}_{k+1} \leftarrow \arg\max_\pi \mathbb{E}_{\mathbf{s}\sim\mathcal{S},\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[\hat{Q}^{k+1}_\theta(\mathbf{s},\mathbf{a}) - \beta\text{Unc}(\mathbf{s},\mathbf{a})\right], \tag{3.27}$$

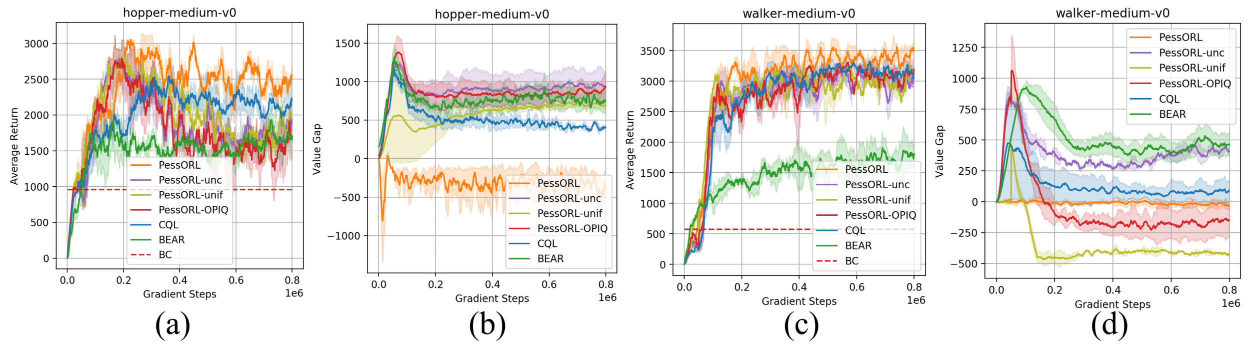


Figure 3.3: (a) and (c): The learning curves in hopper-medium-v0 and walker2d-medium-v0, respectively. (b) and (d): The value gap Δ_k in hopper-medium-v0 and walker2d-medium-v0, respectively.

where the uncertainty term $\text{Unc}(\mathbf{s}, \mathbf{a})$ is similar to the one in the original PessORL, but we no longer take the expectation over the action. Therefore, it becomes $\text{Unc}(\mathbf{s}, \mathbf{a}) = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{f}_{\phi_i}(\mathbf{s}, \mathbf{a}) - \bar{f}_{\phi}(\mathbf{s}, \mathbf{a})\right)^2$. This term evaluates the uncertainty of the dynamics model about a state-action pair. If the sampled state is not in the training dataset, the uncertainty about it will be larger than those in the training data. Therefore, the Q-function is constrained to be lower at out-of-distribution states in this way.

Second, we introduce the variant PessORL-OPIQ. It is inspired by the work in [116], which uses a pessimistic initialization and an optimistic component on the Q-function. Since in the offline settings we need a pessimistic component in the Q-function, we set the PessORL-OPIQ uncertainty term $\text{Unc}(\mathbf{s}, \mathbf{a})$ to be the opposite of the optimistic component in OPIQ, i.e., $\text{Unc}(\mathbf{s}, \mathbf{a}) = -\dfrac{C_{\text{action}}}{(N(\mathbf{s}, \mathbf{a}) + 1)^M}$, where C_{action} and M are hyperparameters in OPIQ, and $N(\mathbf{s}, \mathbf{a})$ is the pseudo count of the state-action pair $(\mathbf{s}, \mathbf{a})$. Here, we use the continuous version of the OPIQ to obtain state-action counts. We adopt the pessimistic initialization from the OPIQ and update the Q function via similar policy evaluation step in Eqn. 3.26, but with an PessORL-OPIQ uncertainty term.

We evaluate the variants PessORL-unc and PessORL-OPIQ on two D4RL MuJoCo environments, and compare their performance to the original algorithm PessORL, CQL, BEAR, and BC based on the average returns on three random seeds. From Fig. 3.3(a) and (c), we can see that the variants PessORL-unc and PessORL-OPIQ both converge to similar returns which are lower than CQL and the original PessORL. Actually, the performance of these two variants are close to each other. We believe the reason is that they both require a super accurate uncertainty estimation and a stringent trade-off factor β. A rough uncertainty estimation may sometimes be harmful to the performance. As we discussed in Sec. 3.6, our uncertainty estimation method based on bootstrapped dynamics models still cannot guarantee an extremely precise estimation. Therefore, directly subtracting the uncertainty term from the learned Q-function to obtain a conservative Q-function may cause the algorithm to perform poorly. On the other side, the original PessORL mitigates the requirement of an accurate uncertainty estimation method because of the additional regularization term in Eqn. 3.23. We also noticed that PessORL-unc and PessORL-OPIQ still outperform the BEAR algorithm. The reason is that we implement PessORL-unc and PessORL-OPIQ on top of CQL, which is a strong offline RL method, so we can attribute most of their good performance to CQL.

From Fig. 3.3(b) and (d), we can see that PessORL-unc maintains a value gap that is among the highest. PessORL-OPIQ is not pessimistic enough as shown in Fig. 3.3(b), but is too pessimistic as shown in Fig. 3.3(d). This indicates that they cannot effectively control the gap and thus cannot assign lower values to out-of-distribution states. The original PessORL algorithm can successfully maintain a value gap that is close to zero, indicating that it shape the value function to be the desired shape.

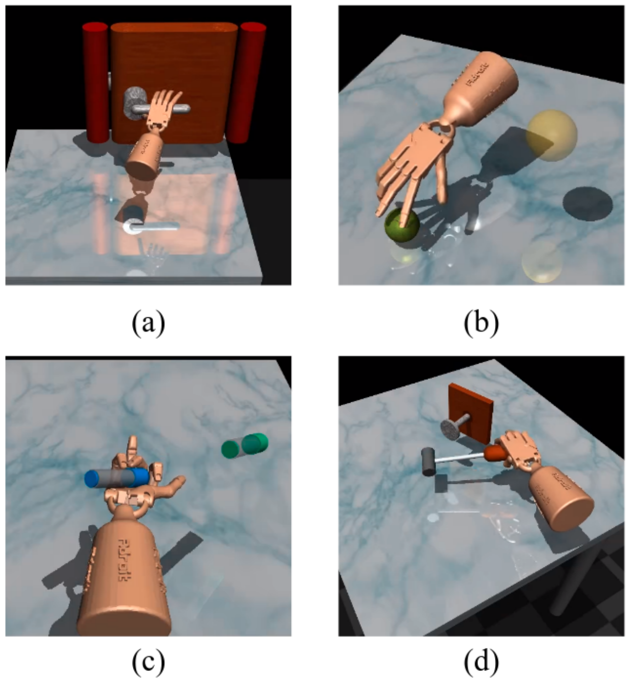


Figure 3.4: Screen shots of Adroit tasks. (a) Door; (b) Relocate; (c) Pen; (d) Hammer.

Generalization to Real Robots

Although the Adroit environments are robotic manipulation simulations, they are considered to be complex enough so that the performance of an offline RL method on these tasks can be viewed as a strong evidence of how the performance will be on real robots. There are actually prior works [114] that use Adroit environments to demonstrate the ability of RL algorithms to adapt to complex tasks. The Adroit environments, as shown in Fig. 3.4, contains four challenging dexterous robot manipulation tasks, which include controlling a 24-DoF simulated manipulator to twirl a pen, open a door, hammer a nail, and relocate a ball. The simulator of Adroit environments provides carefully modeled kinematics, dynamics, and sensing details of the physical hardware to encourage physical realism. The observations contain joint angles, position and orientations of the hand, the object and the target. The actions include the desired position of hand joints. These are all considered to be highly realistic. Therefore, the performance of our proposed PessORL on the Adroit domain compared to other offline RL methods can indicate a good evidence of how it will behave when transferred to real robots.

Besides, our proposed method PessORL matches the simulation results of prior methods that have been demonstrated to work on real robots. Singh et al. [128] showed that CQL can chain behaviors from data and the learned policy can work on a real robot WidowX. From Fig. 3.1, Fig. 3.2, and Tab. 3.1, we can see that PessORL achieves similar or higher performances on all three domains. Therefore, we believe our proposed method should not be too hard to achieve good performance on real robots.

Furthermore, a big advantage of offline reinforcement learning methods is that unlike online RL methods, they are designed to learn policies from pure data and then can be deployed to real robots. Learning directly from previously collected data can dramatically reduce the safety risk in the learning process in safe-critical tasks, such as robotic manipulation and autonomous driving. Actually, it can be a major block for online reinforcement learning to be deployed on real-world scenarios, because online interaction can be impractical in many settings. We show the performance of our proposed PessORL algorithm learned from human demonstrations in Sec. 3.6. These experiments align with the motivation of developing offline reinforcement learning methods, and we believe they can provide a good evidence of the transfer-ability of our method to real-robots.

3.7 Conclusion

We propose a Pessimistic Offline Reinforcement Learning framework to deal with out-of-distribution states. In particular, we add a regularization term in policy evaluation step to shape value function, so that we can improve its extrapolation to OOD states. We also provide theoretical guarantees that the learned pessimistic value function lower bounds the true one and assigns smaller values to OOD states compared to those at in-distribution states. We evaluate the PessORL algorithm on various benchmark tasks, where we show that our method gains better performance by explicitly handling OOD states compared to those methods merely considering OOD actions.

Algorithm 1: Pessimistic Offline Reinforcement Learning (PessORL)

```
1  **Initialize**: A Q network $Q_\theta$ parametrized by $\theta$, A target network $Q_{\bar{\theta}} = Q_\theta$
     parametrized by $\bar{\theta}$, a policy network $\pi_\varphi$ parametrized by $\varphi$, and a bag of dynamics
     models $\{\hat{P}_1, \hat{P}_2, \dots, \hat{P}_n\}$ to detect OOD states;
2  // **Dynamics Models Training** (Models are used by $u_{\pi_\varphi}(\mathbf{s})$ to detect OOD states
     in the policy evaluation step)
3  **for** *step i in range(*0*, M)* **do**
4      Train dynamics models $\{\hat{P}_1, \hat{P}_2, \dots, \hat{P}_n\}$ according to the transitions in the
         dataset $\mathcal{D}$, so that we can later obtain an uncertainty estimation model $u_\pi(\mathbf{s})$ in
         the policy evaluation step;
5  **end**
6  // **Policy Evaluation and Improvement**
7  **for** *step t in range(*0*, N)* **do**
8      Update $Q_\theta$ according to Eqn. 3.23 with learning rate $\epsilon_\theta$ and $u_{\pi_\varphi}(\mathbf{s})$:
9      $\theta_t \leftarrow \theta_{t-1} + \epsilon_\theta \nabla_\theta J(\theta)$ ;
10     Update $\pi_\varphi$ according to the soft actor critic style objective and learning rate $\epsilon_\varphi$:
11     $\varphi_t \leftarrow \varphi_{t-1} + \epsilon_\varphi \mathbb{E}_{\mathbf{s} \sim d^{\pi_\beta}(\mathbf{s}), \mathbf{a} \sim \pi_\varphi(\mathbf{a})}[Q_\theta(\mathbf{s}, \mathbf{a}) - \log \pi_\varphi(\mathbf{a}|\mathbf{s})]$;
12     **if** *t mod target_update* == 0 **then**
13         Soft Update the target network $\bar{\theta}_t \leftarrow (1-\tau)\bar{\theta}_{t-1} + \tau\theta_{t-1}$
14     **end**
15 **end**
```

Table 3.1: Performance on Gym and Adroit Domains

Domain	**Task**	**BC**	**TD3**	**BEAR**	**CQL**	**PessORL**
Gym	hopper-medium	29.71 ± 2.43	0.80 ± 1.11	50.08 ± 4.49	63.04 ± 8.64	$\mathbf{76.39} \pm 4.80$
	walker2d-medium	13.06 ± 1.74	4.91 ± 1.08	34.20 ± 8.57	70.67 ± 0.95	$\mathbf{75.95} \pm 2.96$
	halfcheetah-medium	35.81 ± 0.96	24.60 ± 0.79	41.25 ± 2.45	48.66 ± 0.11	$\mathbf{49.21} \pm 0.59$
	ant-medium	83.42 ± 9.63	-63.55 ± 0.28	-45.78 ± 3.18	51.29 ± 3.92	$\mathbf{85.31} \pm 8.56$
	hopper-medium-expert	85.22 ± 1.38	12.12 ± 0.64	37.67 ± 1.84	105.84 ± 3.62	$\mathbf{112.80} \pm 4.30$
	walker2d-medium-expert	16.12 ± 0.58	4.77 ± 2.06	17.29 ± 2.46	75.27 ± 13.33	$\mathbf{89.67} \pm 6.73$
	halfcheetah-medium-expert	$\mathbf{37.04} \pm 2.77$	-1.41 ± 0.99	-2.93 ± 4.54	19.13 ± 9.81	24.33 ± 12.24
	ant-medium-expert	$\mathbf{66.49} \pm 0.76$	-63.54 ± 0.92	31.31 ± 8.72	34.44 ± 30.36	49.95 ± 16.76
	hopper-random	9.44 ± 1.41	8.47 ± 0.52	9.29 ± 2.58	10.36 ± 3.68	$\mathbf{10.82} \pm 4.11$
	walker2d-random	2.08 ± 0.54	$\mathbf{6.04} \pm 1.08$	0.72 ± 0.82	1.11 ± 5.01	2.66 ± 3.15
	halfcheetah-random	2.25 ± 0.45	27.95 ± 0.59	2.16 ± 0.28	26.62 ± 1.28	$\mathbf{28.58} \pm 0.96$
	ant-random	26.00 ± 0.57	-37.31 ± 1.16	24.05 ± 2.42	26.65 ± 8.65	$\mathbf{27.86} \pm 4.87$
	hopper-expert	109.82 ± 1.44	1.81 ± 1.04	0.78 ± 4.57	104.41 ± 10.26	$\mathbf{110.67} \pm 8.60$
	walker2d-expert	60.66 ± 2.26	-0.95 ± 0.58	21.37 ± 3.52	105.55 ± 2.91	$\mathbf{109.37} \pm 4.94$
	halfcheetah-expert	$\mathbf{94.22} \pm 1.09$	-1.40 ± 2.85	13.55 ± 7.67	80.19 ± 14.55	71.33 ± 20.16
	ant-expert	$\mathbf{73.57} \pm 3.28$	55.07 ± 1.99	44.65 ± 10.39	59.04 ± 20.66	60.54 ± 17.83
Adroit	pen-human	34.46 ± 0.83	-3.83 ± 0.43	35.62 ± 1.34	54.49 ± 7.58	$\mathbf{63.07} \pm 4.36$
	door-human	1.46 ± 0.06	-0.19 ± 0.01	-0.33 ± 0.05	1.86 ± 0.29	$\mathbf{2.28} \pm 0.14$
	hammer-human	1.35 ± 0.09	0.26 ± 0.12	0.46 ± 0.03	3.91 ± 0.34	$\mathbf{4.24} \pm 0.28$
	relocate-human	0.04 ± 0.02	-0.32 ± 0.03	-0.30 ± 0.10	0.15 ± 0.04	$\mathbf{0.33} \pm 0.12$
	pen-cloned	23.58 ± 1.14	-3.91 ± 0.36	28.92 ± 9.62	35.23 ± 11.03	$\mathbf{39.02} \pm 9.25$
	door-cloned	0.15 ± 0.08	-0.33 ± 0.01	-0.16 ± 0.04	$\mathbf{1.72} \pm 0.14$	1.69 ± 0.35
	hammer-cloned	0.40 ± 0.07	0.25 ± 0.04	0.21 ± 0.34	0.53 ± 0.52	$\mathbf{0.95} \pm 0.38$
	relocate-cloned	-0.24 ± 0.11	**-0.14** ± 0.08	-0.23 ± 0.13	-0.28 ± 0.57	-0.26 ± 0.24

Part II

Hierarchical Planning Based on Reinforcement Learning

Chapter 4

A Safe Hierarchical Planning Framework for Complex Driving Scenarios based on Reinforcement Learning

Autonomous vehicles need to handle various traffic conditions and make safe and efficient decisions and maneuvers. However, on the one hand, a single optimization/sampling-based motion planner cannot efficiently generate safe trajectories in real time, particularly when there are many interactive vehicles near by. On the other hand, end-to-end learning methods cannot assure the safety of the outcomes. To address this challenge, we propose a hierarchical behavior planning framework with a set of low-level safe controllers and a high-level reinforcement learning algorithm (H-CtRL) as a coordinator for the low-level controllers. Safety is guaranteed by the low-level optimization/sampling-based controllers, while the high-level reinforcement learning algorithm makes H-CtRL an adaptive and efficient behavior planner. To train and test our proposed algorithm, we built a simulator that can reproduce traffic scenes using real-world datasets. The proposed H-CtRL is proved to be effective in various realistic simulation scenarios, with satisfying performance in terms of both safety and efficiency.

4.1 Introduction

Recently the field of autonomous driving has witnessed rapid development. A number of novel behavior planning algorithms have been proposed, many of which are awesome achievements. However, it is almost impossible for a single behavior planner to drive through so many different real and complex scenarios. For example, one needs to ride in an autonomous car in his everyday life. The autonomous car should be able to drive both in cities and on highways, which include various intersections, roundabouts, and merging and following sce-

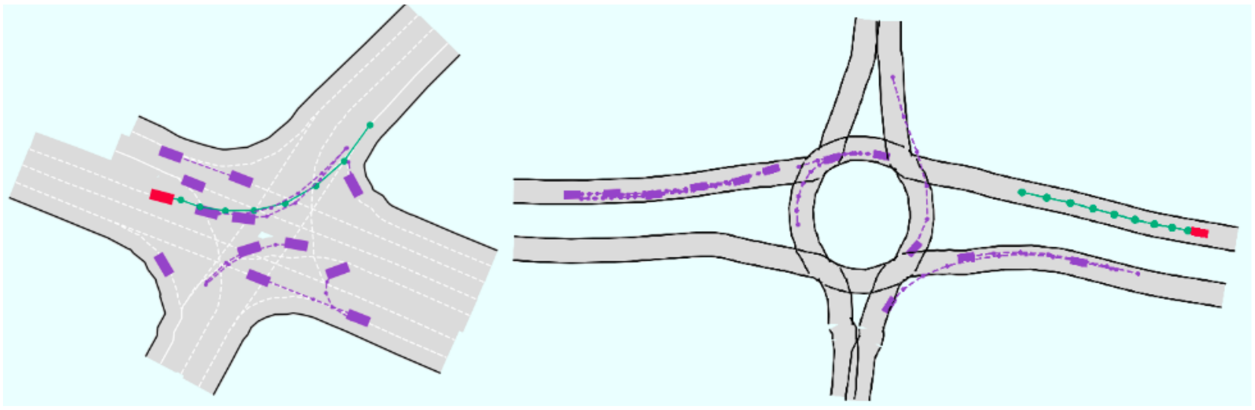


Figure 4.1: Various complex real-world traffic conditions in which autonomous vehicles (red) must drive safely and efficiently.

narios (as shown in Figure 4.1). One single planning algorithm, such as an optimization-based planner, may never find its solution in each complex traffic condition. The hyperparameters in each planner or policy may work well in intersection scenarios, but it is very likely to fail in highway merging scenarios, because the optimal driving settings in each scenario, e.g. the optimal speed and spacing, are so difficult to adjust with only one fixed planner.

The most difficult part in this complex driving problem lies in the rapidly changing environment. The road curvature may change, and the number of obstacles may vary in each of these self-driving scenarios. For example, when an autonomous vehicle is trying to make a left turn at an intersection, it has to consider the number of oncoming vehicles, the actual speed of them, and their distance to the intersection. However, when the self-driving vehicle is finding its way to merge to the next lane on the highway, although it also has to analyze similar information, it needs to do it in a whole different larger scale for distance, velocity, etc. We therefore have to find an adaptive policy that can make high-quality decisions in various scenarios.

Each behavior and motion planner from existing work has its own advantages and disadvantages. For example, learning-based methods can recognize the specific properties of different scenarios with less effort, but it is hard to interpret the results and generalize them to other scenarios. On the other hand, non-learning-based methods can ensure the safety of the agent and deal with many similar cases without too much modification, but it is difficult to tune the hyperparameters (e.g., the detailed cost function of motion planning) to be compatible for various traffic scenarios with different number of interactive agents. Therefore, based on the analysis of these limitations, we seek to design a hierarchical model to exploit the advantages of both reinforcement learning and non-learning-based policies.

In this chapter, we make the following contributions:

- We propose a **H**ierarchical behavior planning framework with low-level safe **C**on**t**rollers and a high-level **R**einforcement **L**earning **(H-CtRL)**. It is an adaptive behavior planner that can make high-quality decisions in many complex traffic scenarios.

- A simulator that could reproduce traffic scenes from real-world traffic datasets is constructed, so that the proposed method can be trained and tested in realistic scenarios.

- We test the proposed method H-CtRL in real-world traffic conditions, and it was proved to be capable of handling different planning tasks in various scenarios.

4.2 Related Work

Non-Learning Based Methods

Non-learning-based decision-making modules are considered to be safe-guaranteed and interpretable [95, 103]. But these planners tends to be too conservative. In [106], this problem was addressed by making the autonomous agent more cognizant of and reactive to obstacles. The authors of [121] proposed another framework leveraging effects on human actions to make it more interactive. However, the methods aforementioned suffer more or less from their long computation time. Constrained Iterative Linear Quadratic Regulator [18] significantly reduced the operation time while preserving the safety. We try to inherit the safe guarantees and fast operation time from these methods, and select non-learning-based planners as our low-level safe controllers.

Supervised Learning

The application of supervised learning on autonomous driving dates back to the work in [109]. The authors of [9] then learned a map from raw pixels directly to steering commands, where the concept of imitation learning (IL) began to surface. A more robust perception-action model was developed in [149]. To enhance the safety of IL, [131] proposed a hierarchical framework which utilized a high-level IL policy and a low-level MPC controller to improve efficiency and safety. Similarly, to make IL generalizable and deal with complex urban scenarios, the authors of [20] learned policies from offline connected driving data, and integrated a safety controller at test time.

Reinforcement Learning

Reinforcement learning (RL) has also been extensively explored in autonomous driving. The algorithm in [122] adopted Recurrent Neural Networks for information integration, and learned an effective driving policy on simulators. The work in [104] developed a realistic translation network to make sim2real possible. [11, 19] developed robust policies to make self-driving cars capable of driving through complex urban scenarios. One can also consider to incorporate prediction models [81, 79] to build model-based RL planners. There are also multi agent RL methods [26] available to be applied on autonomous vehicles. We believe RL is a suitable high-level policy candidate. It can learn from experience to know which low-level controller is the most suitable at a specific time step.

Hierarchical Reinforcement Learning (HRL) can make the learning process more sample-efficient. The idea is to reuse the well trained network of one sub-goal on other similar tasks in HRL [112, 100].

There are also many variants of HRL. The work in [144] integrated a sampling-based motion planner with a high-level RL policy, which can solve long horizon problems. Similarly, [54] combined deep reinforcement learning with Monte Carlo sampling to achieve tactical decision making for autonomous vehicles. Authors of [135] developed SAVED which augmented the safety of model-based reinforcement learning with Value estimation from demonstrations. Only to plan in normal scenarios is not enough for reliable self-driving cars, so the authors of [16] developed a hybrid method with RL and IL policies to plan safely in near accident scenarios. The authors of [77] proposed an attention-based architecture that can deal with a varying number of obstacles and the interaction in between.

We adopted the basic ideas to reuse low-level controllers, and aimed to design a novel planning module that works in various traffic conditions. The high-level RL was trained to recognize and react to different environments, while the low-level conventional controllers fulfill the goals sent from the high-level RL and guarantee the safety in the same time.

4.3 Problem Statement

Throughout the chapter, we focus on the behavior planning problem in different complex urban traffic scenarios. There is one ego agent and many other obstacle cars in the environment. Each of them has its own behavior pattern. Thus, we need a mechanism to model the evolution of each scene and the interactions among agents. We formulate the problem as a Partially Observable Markov Decision Process (POMDP). A POMDP can be defined as a tuple: $<\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{Z}, \mathcal{R}, \gamma>$. $\mathcal{S}$ denotes the state space and $s \in \mathcal{S}$ is a state of the environment. $\mathcal{A}$ is defined to be the action space and $a \in \mathcal{A}$ is an action taken by the ego agent. $o \in \mathcal{O}$ is an observation received by the ego agent. The transition model $\mathcal{T}(s, a, s')$ is the probability of the transition from a current state - action pair (s, a) to s' at the next time step. $\mathcal{Z}(s, o)$ denotes the transition model, which calculates the probability of ending in the observation o given a state s. The reward function is defined by $\mathcal{R}(s, a)$, which yields a specific reward via a state - action pair (s, a). The discount factor is denoted by γ. The overall objective is to maximize the expected discount reward and find the corresponding optimal policy

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s^t, \pi(o^t))\right] \tag{4.1}$$

where s^t, o^t are the state and the observation at timestep t of the environment, respectively.

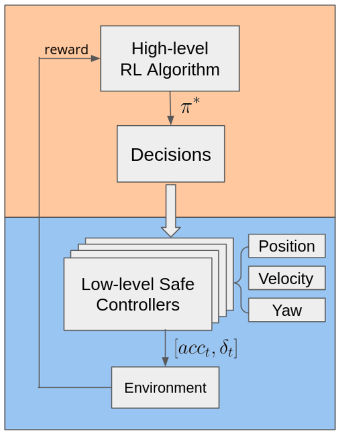


Figure 4.2: The hierarchical framework with low-level safe controllers and a high-level RL algorithm.

4.4 Method

H-CtRL Framework

We propose to solve the POMDP aforementioned with a hierarchical behavior planning framework H-CtRL (shown in Figure 4.2). It can be viewed as an integrated solver for the POMDP. We input the current state of the environment into the framework and then it outputs actions to be executed by the ego agent. The hierarchical framework consists of a collection of low-level controllers with safety constraints and a high-level Reinforcement Learning policy to manage them all. We aim to find an optimal high-level policy π^* that can take advantage of one controller at a specific time step when doing behavior planning for the autonomous agent.

In details, the state s_t of the problem at the time step t is designed to be the low-dimensional states of all agents presented in the environment at t, namely,

$$\begin{aligned} s_t &= \left[s_t^1 \;\; || \;\; s_t^2 \;\; || \;\; \ldots \;\; || \;\; s_t^m\right] \\ s_t^i &= \left[x_t^i \quad y_t^i \quad v_t^i \quad \theta^i\right], \; i \in [1, m] \end{aligned} \tag{4.2}$$

where $s_t^i, \; i \in [0, m]$ is the low-dimensional state of the i-th agent in the environment at time step t (note that s_t^0 denotes the state of the ego agent), and s_t is the state of the environment. The operator $||$ here is the concatenation operator. The action at time step t is denoted by $a_t = [acc_t \;\; \delta_t]$ where acc_t and δ_t are the acceleration and the steering angle of the ego agent, respectively. We choose the states of the k nearest neighbors around the ego vehicle at time step t to be the observation, namely, $o_t = [o_t^1 \;\; || \;\; o_t^2 \;\; || \;\; \ldots \;\; || \;\; o_t^k]$ where $o_t^i, \; i \in [1, k]$ is the state of the i-th nearest obstacle around the ego car. We refer [69] and select the bicycle model as the transition model for the ego agent. As for the other vehicles

Algorithm 2: H-CtRL Training Algorithm

```
1  Initialize: A simulation environment $env$ and get an initial observation $o_t$. A policy
     and a target Q-network with weights $\theta$ and $\theta^-$. Set $\theta^- \leftarrow \theta$. an empty reply buffer
     $\mathcal{B}$ with a maximum length of $l_B$;
2  for $h \leftarrow 0$ to $N$ steps do
3      Select an action $a_h \leftarrow \arg\max_{a_h} Q_\theta(o_h, a_h)$ according to the $\epsilon - greedy$;
4      Given $o_h$, the chosen low-level controller corresponding to $a_h$ acts $p$ timesteps in
         $env$ to get the next observation $o_{h+1}$ and the reward $r_h$;
5      Push the transition $\mathcal{T} = \begin{bmatrix} o_h & a_h & r_h & o_{h+1} \end{bmatrix}$ into $\mathcal{B}$;
6      Update the weights $\theta$ of the policy Q-network using the replay buffer $\mathcal{B}$;
7      if $h$ mod $target\_update\_frequency == 0$ then
8          $\theta^- \leftarrow \theta.copy()$
9      end
10     if the episode is done then
11         Record the cumulative reward in this episode;
12         Reset $env$ and get an initial observation $o_{t+1}$;
13     end
14 end
```

in the scene, their states would evolve according to the definition by the environment or the simulator.

Low-Level Safe Controllers consist a set of n non-learning-based controllers, denoted by $\{f_1, f_2, \ldots, f_n\}$. They are like the workers within the hierarchical framework, who are responsible for specific tasks assigned by high-level coordinators. Each $f_i,\ i \in [1, n]$ has its own behavior pattern, either cooperative or selfish, either defensive or aggressive. Although they can lead to different driving strategies, they all have their own safety constraints when performing motion planning. For example, sampling-based planners would assign little probability to the area where an accident is more likely to happen, and optimization-based controllers would have huge cost in the objective function when the output lands in the dangerous area. Since the chosen low-level controller is the one whose action directly affects the evolution of the environment, these safety constraints of the low-level controllers are generally a good property that guarantees the safety of the whole hierarchical behavior planning framework. Specifically, the low-level controller takes in the observation o_t and gives back an action a_t at the time step t.

The High-Level Reinforcement Learning policy is the coordinator in this framework. It makes observations from the environment and gets to choose one of the low-level controllers that is the most suitable given the current observation. Therefore, the action space of the high-level RL is reduced and discretized from the original continuous space to a finite set of low-level controller's id $\{1, 2, \ldots, n\}$. The actions of RL can be viewed as intermediate actions within the hierarchical framework, whereas the final output actions that directly interact and affect the environment are the outputs of low-level controllers. Here, we should note that to reduce the cardinality of the actions within one episode and improve the stability of the algorithm, the high-level RL switches its choice of low-level controllers every p time steps, which means within the p time steps of the environment, only one chosen low-level controller would plan trajectories consistently and would not be disabled by the RL. We therefore introduce the new timestep h for the high-level RL: the original time step $t = h \cdot p$ should coincide with the time step h in RL.

The state and the observation at each time step of the RL problem are defined accordingly as s_h and o_h as aforementioned. One way to get s_h and o_h is to convert the time scale in RL back to the original scale, thus s_h and o_h should be the state and the observation at time step $t = h \cdot p$ in the environment. The transition model of RL is also defined within the new time scale, namely, $\mathcal{T}(s_h, a_h, s_{h+1})$, where s_{h+1} is the next state of s_h after applying p actions by the low-level controller corresponding to a_h. Because of the existence of low-level controllers, we no longer need to worry about various tedious design details of the reward function $\mathcal{R}(s_h, a_h)$, and can simply adopt a very high-level one that encourages the completion of the planning task as fast as possible without any collision. In detail, the ego agent receives a positive reward that is proportion to its progress along its reference trajectory and a negative reward if the episode terminates early because of collision or low-level controller failure or other factors.

Generally, the high-level RL can be trained using any model-free RL algorithms. In this chapter, we choose to use Double Deep Q-Network (DDQN) in [53] to learn our high-level RL policy, for it is more stable and has less variance. The pseudo-code for the hierarchical planning framework is shown in algorithm 2.

4.5 Experiments

Simulator

We constructed our own simulation environment based on the INTERACTION Dataset [156] and the OpenAI GYM toolkit. The road maps in the simulator were loaded from the INTERACTION Dataset map collections, which contained various real-world traffic scenarios recorded from many different countries. After the simulator finished constructing the road map, we specify an initial timestep from where vehicles data were loaded. The states of these vehicles other than the ego agent were all loaded from the dataset at each time step. Since these vehicle data were all collected from real-word traffics, we could simulate relatively realistic traffic conditions. The ego agent in the simulator would then be our self-driving car equipped with H-CtRL. The transition model of the ego agent was the bicycle model. When running experiments in the simulator, we input into it one low-level action, $a_t = [acc_t,\ \delta_t]$, and then it would take one step and output a reward, an observation, and a boolean indicating whether the episode had terminated, according to the bicycle model and the dataset.

Scenarios

We consider two road maps from the INTERACTION dataset, and design different driving tasks in each of them.

- **TC_BGR_Intersection_VA (VA)**. It is a map of a busy and complex intersection, which makes it difficult for the ego agent to avoid collisions.
- **DR_USA_Roundabout_SR (SR)**. The map is collected from a real-world roundabout. The map is bigger than the previous intersection map and thus is difficult to navigate through.

Since there are four directions in both scenarios, we design similar tasks in each of them. The ego agent should navigate through the traffic safely to make unprotected left turns, unprotected right turns, and straight crossings. By unprotected left or right turns, we mean that one must yield to other vehicles when turning left on green lights and turning right on red lights according to the traffic rules.

Low-level Controllers

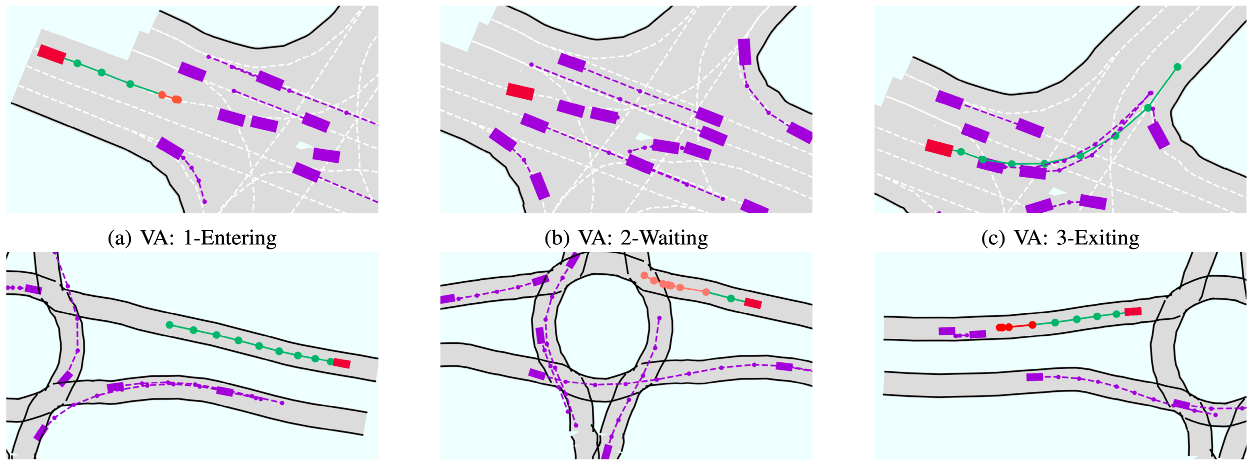
Generally speaking, we can choose any mature non-learning-based planner to make the proposed hierarchical framework inclusive and powerful. In this chapter, we consider $n = 9$ different Constrained Iterative Linear Quadratic Regulators (CILQR) [18] as the set of low-level controllers. The objective of CILQR is to find an optimal control sequence, namely, an optimal action sequence a^* given an initial observation o_0 that minimizes a cost function:

$$\begin{aligned} a^*, o^* = \arg\min_{a,o} & \left\{ \phi(o_N) + \sum_{t=0}^{N-1} L(o_t, a_t) \right\} \\ \text{s.t.} \quad & o_{t+1} = f(o_t, a_t), \quad t = 0, 1, \ldots, N-1 \\ & g_t(o_t, a_t) < 0, \quad t = 0, 1, \ldots, N-1 \\ & g_N(o_N) < 0 \end{aligned} \tag{4.3}$$

where N is the planning horizon, $L(\cdot)$ and $\phi(\cdot)$ are the cost functions, $f(\cdot)$ is the transition model, and $g_t(\cdot)$'s are the safety and dynamics constraints.

From a theoretical point of view, Chen *et al* proved in Theorem 1 in [18] that for the problem in Equation 4.3, the output trajectories $\{o_t^{(k)}, a_t^{(k)}\}$ at the $k-$th step will converge to a local optimum as $k \to 0$ when using the CILQR algorithm. Compared to other non-learning based methods, CILQR solves the optimal control problem with non-convex constraints and non-linear system dynamics much faster with a guarantee to converge. Learning based methods do not introduce constraints on dynamics and have no guarantee of a convergence either.

It has been tested with on-road driving scenarios and is proved to be able to avoid obstacles successfully. However, the main drawback of CILQR is that it tends to be very


(a) VA: 1-Entering (b) VA: 2-Waiting (c) VA: 3-Exiting

(d) SR: 1-Entering (e) SR: 2-Waiting (f) SR: 3-Following

Figure 4.3: The planned trajectories in both maps. The ego car is always red and the obstacles are always purple. The future trajectories for the next 10 high-level timestep h are plotted using a line with markers on it. A green line means H-CtRL chooses a low-level CILQR planner with high reference speed, while a red line means H-CtRL chooses one with low reference speed. The darker the red, the lower the speed.

aggressive if its reference speed is too fast. For example, when the ego agent is following slow traffic in an urban scenario, it always tries to pass the cars in the front whenever it finds a gap. This maneuver style may cause serious problems, because a sudden move is highly likely to result in collisions in such dense traffic with many occlusions. The dangerous decision is mainly because of the high reference speed that the controller tempts to track. Since the objective function penalizes its deviation from the reference trajectory, it is willing to sacrifice the safety to bypass the obstacles.

Therefore, we seek to apply the high-level RL policy to choose the most suitable reference and the most appropriate setting for low-level controllers. We design a fixed finite set of candidate reference speed for the high-level RL to choose. The set includes 9 possible discrete speeds: $v_{\text{ref}} \in \{0, 2, 3, 4, 5, 6, 7, 8, 9\}(m/s)$. Each reference speed corresponds to a different CILQR controller that has a different behavior pattern. For example, the one with the reference speed $v_{\text{ref}} = 0m/s$ is the most conservative one, because it would yield to any obstacle in the environment, whereas the one with $v_{\text{ref}} = 9m/s$ is the most aggressive one, for it would tries its best to track the high reference speed and the safety would be compromised. The high-level RL aims to balance between the safety and the passing time given the current observations, so as to make the ego agent capable of handling various complex scenarios.

Baseline Methods

We compared the following policies in the experiments:

- **CILQR#3**. The third low-level CILQR controller with a reference speed of $3m/s$. No high-level RL policy is used.
- **CILQR#9**. The ninth low-level CILQR controller with a reference speed of $9m/s$. No high-level RL policy is used.
- **Random**. The hierarchical framework with the high-level RL is replaced by a random sampler, which chooses low-level controllers randomly.
- **H-CtRL**. Our proposed hierarchical behavior planning framework.

It would be tedious to compare and list results of all low-level CILQR controllers, so we only choose two representative low-level controllers here. CILQR#3 plans trajectory that tracks a low reference speed, which will results in a rather conservative driving strategy, whereas CILQR#9 is just the opposite resulting in an aggressive driving strategy.

Implementation Details

We trained and tested the RL in our proposed hierarchical framework in two maps separately. At the beginning of each episode, we initialized the position of the ego vehicle at the edge of the road map, perturbed by a Gaussian noise. The initial timestep to load obstacles from the dataset was randomly sampled from 600 to 900 original timestep in VA and from 100 to 400 in SR. Each timestep in the simulator as well as CILQRs lasts 0.1s, whereas each timestep of the high-level RL lasts 1.0s with $p = 10$.

The goal of each episode was also chosen randomly, either to turn left, turn right, or to go straight. The observation that was fed into the high-level RL was the observation given by the simulator plus the goal of each episode. The RL policy was represented by a neural network with two fully connected hidden layers. According to the high-level decision, the same observation was then fed into the low-level controller to plan executable actions in the simulator for the ego agent.

4.6 Results

Statistics

We ran each policy for 100 episodes in VA and SR separately, and compared the average episode return, the collision rate, and the completion rate within a 50s time limit.

As shown in Table 4.1, the proposed H-CtRL has the best performance in terms of the average episode reward. It is much higher than CILQR#9 and Random, while CILQR#3 is close to it. Considering the experiment setting where initial states of the environment are sampled randomly among a wide range, we can safely conclude that our proposed H-CtRL

Table 4.1: Average episode returns, collision rate, and the completion rate within 50s time limit based on 100 episodes in each map.

Method	Aver. Epi. Return	Collision Rate	Completion Rate
Intersection (VA)			
CILQR#3	80.06	0.07	0.24
CILQR#9	37.84	0.59	0.09
Random	51.23	0.36	0.17
H-CtRL	86.59	0.10	0.85
Roundabout (SR)			
CILQR#3	73.52	0.05	0.17
CILQR#9	48.64	0.47	0.22
Random	55.13	0.32	0.28
H-CtRL	90.29	0.08	0.91

is able to handle various situations better than each individual low-level planners. It also implies that H-CtRL is better than a random high-level switching policy, so the high-level RL successfully learned useful skills to navigate through various complex urban traffics.

When looking into the collision rate, CILQR#3 performs the best, following by H-CtRL. CILQR#9 is the most aggressive policy, as is implied by its high reference speed. When we set a time limit of 50 seconds for each episode, only the proposed H-CtRL has the ability to finish the task in both maps. CILQR#3 is too conservative and drives at a low speed, making it almost impossible to finish the task in time. CILQR#9 is just the opposite of CILQR#3. It drives too fast to recognize danger and to avoid collisions in time. The random switching policy fails to reach a high completion rate because of a mixture of the reasons aforementioned. If we look into the collision rate and the completion rate together, we can conclude that H-CtRL makes a good balance between the safety and the operation time.

Visualization

To show the details of what happened in both VA and SR, we visualized one representative episode in each map.

Figure 4.3(a)-(c) are the visualization for an episode in VA, where the ego car was trying to make a left turn when the traffic light was green. According to traffic rules, it must yield to oncoming cars from across the road. As we can see in Figure 4.3(a), the high-level RL first chose to drive at a high speed of approximately $6m/s$ (as shown by the green markers) into the entrance of the intersection. Then it decided to slow down in front of the intersection (as shown by the orange markers). When cars continued to come from the opposite direction, the high-level RL planned a perfect stop to stay put (as shown in Figure 4.3(b)). After all cars finished crossing the intersection, It decided to accelerate and to exit the intersection as fast as possible (as shown in Figure 4.3(c)).

One episode in SR was visualized in Figure 4.3(d)-(e). In this episode, the ego agent was trying to go straight across the roundabout. When it was approaching the roundabout, the high-level RL chose to drive at a high speed as usual (implied by the green markers in Figure 4.3(d)). When it was about to enter the roundabout, the high-level RL decided to slow down (shown in Figure 4.3(e)) so that it could observe the surroundings and could avoid collisions if necessary. After it exited the roundabout (shown in Figure 4.3(f)), it first accelerated toward the goal position. But there were several obstacles driving slowly in the front, so it chose a low-level controller with a very low reference speed ($v_{\text{ref}} = 2m/s$). Then the low-level CILQR controller helped the ego agent to slow down and avoid collisions.

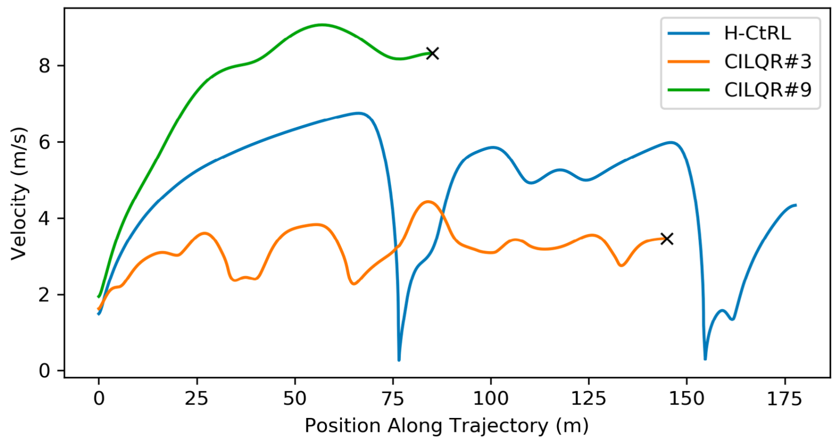
We visualized the velocity and the task progress of H-CtRL for the SR episode described above, and compared them with those of CILQR#3 and CILQR#9 in Figure 4.4. As we can see, neither CILQR#3 nor CILQR#9 managed to finish the task before the time was up. The ego agent with CILQR#9 collided with obstacles in the episode, whereas the one with CILQR#3 did not manage to finish the task before the time was up. Only the agent with H-CtRL successfully finished this episode. We can therefore conclude that our proposed H-CtRL has the ability to handle these complex urban traffic conditions safely and efficiently.

Failure Cases

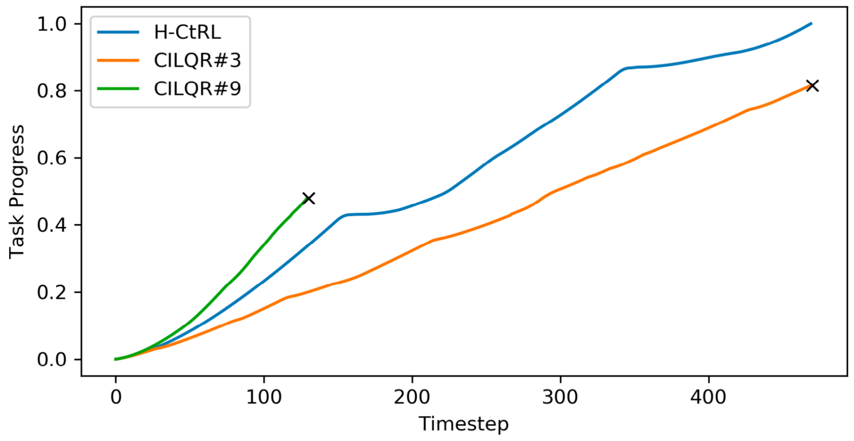
When training and testing our proposed method, we discovered that the ego agent braked really hard if it observed obstacles in the front or the front vehicle slowed down. We believed that this is caused by the low-level CILQR controllers, which only considered the safety constraint without any optimization on the comfort. Generally speaking, our proposed hierarchical framework has the ability to adopt many low-level planners. Therefore, the future work is to add more low-level planners into the framework, and to train them all together to get a more powerful and generalizable behavior planner.

4.7 Conclusion

In this chapter, we proposed a general behavior planner for autonomous vehicles based on reinforcement learning and safe motion planners. By combining the power of low-level safe controllers with a high-level reinforcement learning coordinator, various complex urban traffic conditions can be handled via this general framework. We built a simulator that can reproduce scenarios according to real-world traffic dataset. The proposed algorithm was trained and tested based on such real traffic data. Compared to other baseline methods, the proposed framework achieved both high completion rate and low collision rate, verifying its ability to handle various traffic scenarios with satisfying performance on both safety and efficiency.



(a) The velocity vs. the position along the trajectory of the ego agent.



(b) The task progress vs. the timestep of the ego agent.

Figure 4.4: A visualization of the velocity and the task progress of the ego agent driving in SR. The proposed H-CtRL was compared to CILQR#3 (with $v_{\text{ref}} = 3m/s$) and CILQR#9 (with $v_{\text{ref}} = 9m/s$). The black cross means that the episode was terminated earlier without reaching the goal position.

Chapter 5

Hierarchical Planning Through Goal-Conditioned Offline Reinforcement Learning

Offline Reinforcement learning (RL) has shown potent in many safe-critical tasks in robotics where exploration is risky and expensive. However, it still struggles to acquire skills in temporally extended tasks. In this chapter, we study the problem of offline RL for temporally extended tasks. We propose a hierarchical planning framework, consisting of a low-level goal-conditioned RL policy and a high-level goal planner. The low-level policy is trained via offline RL. We improve the offline training to deal with out-of-distribution goals by a perturbed goal sampling process. The high-level planner selects intermediate sub-goals by taking advantages of model-based planning methods. It plans over future sub-goal sequences based on the learned value function of the low-level policy. We adopt a Conditional Variational Autoencoder to sample meaningful high-dimensional sub-goal candidates and to solve the high-level long-term strategy optimization problem. We evaluate our proposed method in long-horizon driving and robot navigation tasks. Experiments show that our method outperforms baselines with different hierarchical designs and other regular planners without hierarchy in these complex tasks.

5.1 Introduction

Reinforcement learning (RL) has been widely applied for a broad range of tasks in robotics. However, it remains challenging to solve those complex tasks with extended temporal duration with RL, especially for safety-critical applications where exploration is risky and expensive (e.g., autonomous driving). To avoid risky exploration, offline RL has drawn growing research attention recently [44, 74, 85], for its ability to train RL policies from static offline datasets. Many prior works have investigated offline RL algorithms in the area of robotics and autonomous driving [73, 128]. However, no prior works have deliberately investigated offline RL algorithms for temporally extended tasks. In this work, we aim to study this problem, which is an important step towards the applications of RL to navigate intelligent agents in complex and safety-critical environment.

To deal with extended temporal duration, a promising solution is to adopt a hierarchical framework, with a low-level policy controlling the agent to accomplish short-term tasks, while receiving supervision from a high-level module reasoning about long-term strategy. The high-level module could be a model-free RL policy [34, 96, 83] or a planning module [98]. We are particularly interested in [98], which combines a low-level goal-conditioned RL policy [65, 123] and a high-level model-based planning module guiding the goal-reaching policy with a sub-goal sequence. The high-level planning takes the advantages of model-based approaches [111, 8] to handle long-horizon tasks by composing behaviors. We found it particularly suitable for offline setting because it only requires learning a short-horizon goal-reaching policy from offline dataset, which is much easier and more data-efficient than directly learning an end-to-end policy solving the entire task.

In this work, we propose a hierarchical planning framework through goal-conditioned offline reinforcement learning. We leverage Hindsight Experience Replay (HER) [6] to synthesize goal-conditioned episodes from static datasets. Afterwards, we train the policy with state-of-the-art model-free offline RL algorithms. To deal with the distributional shift caused by out-of-distribution (OOD) goals, we propose to generate noisy unreachable goals by a perturbed goal sampling process. By incorporating those noisy goals into training, it informs the high-level planner to avoid OOD goals during online execution. The high-level sub-goal planner plans over intermediate sub-goals for the low-level policy. Specifically, it solves an optimization problem based on the goal-conditioned value function of the low-level policy online. To ensure efficient online computation, we train a Conditional Variational Autoencoder (CVAE) [130] to propose goal sequence candidates that are feasible and reasonable.

We evaluate our proposed framework for planning in various domains, specifically, robot navigation tasks and driving scenarios. Experiment results show that our framework is superior to the policy trained by regular offline reinforcement learning methods without a hierarchical design. The trained value function can quantify the quality of different goal candidates in the high-level goal planner. Also, our framework can be generalized to complex driving scenes with lower collision rates than baselines.

5.2 Related Work

Hierarchical framework has shown promising in dealing with long-horizon tasks. Conventional hierarchical RL decomposed the action space directly [39, 25, 84]. Recent works have proposed different architecture designs with novel high-level and low-level components, e.g., high-level graph search-based planner [41, 80, 150], two-phase training [93], mixed imitation learning and reinforcement learning policies [49, 92], and high-level latent primitive extraction [4]. They typically require training multiple policies online, or the task to be repetitive. In contrast, our proposed framework only requires training one RL low-level policy from static offline data. Also, the value function used by the high-level planner corresponds to the short Markov decision process (MDP) between consecutive sub-goals instead of the whole MDP as in previous works [92]. Consequently, our method only requires estimation of the expected future return over the short horizon between sub-goals, instead of the entire episode length. Plus, our method can consider multiple look-ahead timesteps instead of one as in previous works [92, 4], increasing the ability to reason about long-term strategies.

Our high-level planner is similar to the one in [98], which is developed for a finite-horizon goal-conditioned MDP with a single objective of goal reaching. In contrast, ours generalizes it to MDPs that are not goal-conditioned, so that we can consider criteria besides reachability (e.g., comfort, safety). In [98], a variational autoencoder (VAE) is adopted to model high-dimensional image observation for efficient sub-goal sampling during online optimization. The sub-goals at different timesteps are modeled independently, inducing unnecessary noise into the high-level optimization. We instead use a CVAE to sequentially sample sub-goal sequences which improves the optimization performance. Most importantly, our framework is developed under the offline setting, whereas the planner in [98] requires learning the low-level policy and value function with online exploration.

5.3 Problem Formulation

We consider a MDP represented by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r_{\text{env}}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the transition function, r_{env} is the reward function, and γ is the discount factor. An important component when using RL to solve a MDP is the value function under a policy. Value function estimation tends to be accurate when predicting at points in the neighborhood of the present observation [61]. It is then reasonable to break a big task into small pieces, so that the policy only needs the corresponding value function to predict values at close and accurate goal points. Based on this insight, we solve the MDP with a hierarchical policy where a high-level planner is responsible for sub-goal selection, and a low-level goal-conditioned policy is responsible for generating executable actions.

The goal planner operates at a coarse time scale, generating a sub-goal at every N time steps. We define the time steps at which the goal planner outputs a sub-goal as *high-level time steps*, denoted by t_i, with $t_{i+1} - t_i = N$. Given a planned sub-goal at t_i, the low-level policy generates actions at each time step between t_i and t_{i+1}, controlling the agent to reach the desired sub-goal at t_{i+1}. Formally, the low-level goal-conditioned policy is denoted by $\pi(\mathbf{a}_{t_{i,j}}|\mathbf{s}_{t_{i,j}}, \mathbf{g}_{t_{i+1}})$ for $j = 0, 1, ..., N-1$, where $t_i = t_{i,0}$, and $\mathbf{g}_{t_{i+1}}$ is the sub-goal for t_{i+1} generated at the previous high-level time step. The low-level policy essentially solves a short-horizon goal-reaching MDP defined between two consecutive high-level time steps. In the next section, we will show that the planning problem can be formulated as an optimization problem with an objective function defined with the value function of the goal-reaching MDP.

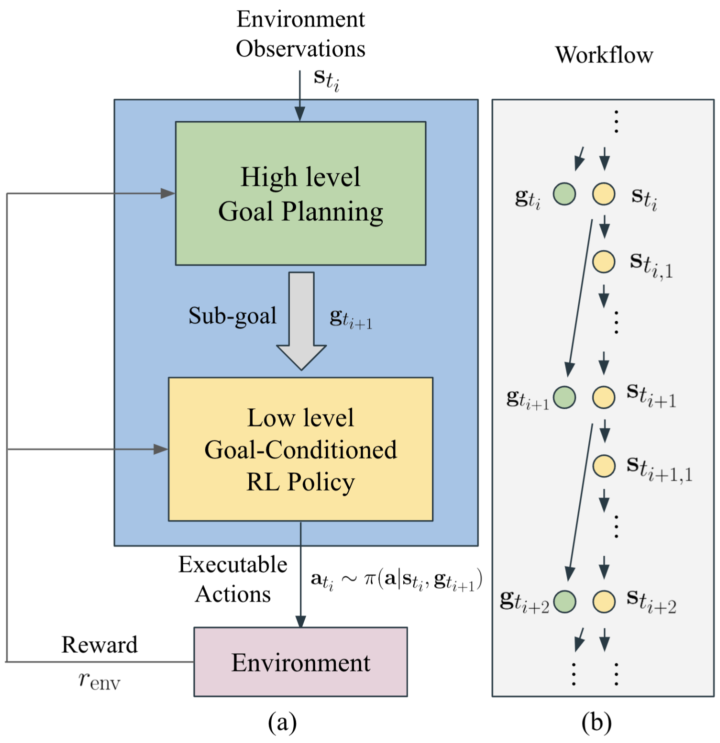


Figure 5.1: (a) The hierarchical goal-conditioned offline reinforcement learning framework. (b) The workflow of the proposed framework.

5.4 Hierarchical Goal-Conditioned Offline Reinforcement Learning

We now present the proposed hierarchical planning framework shown in Fig. 5.1 by answering the following questions: 1) How the planner generates sub-goals; 2) How to combine the sub-goal planner and the goal-conditioned policies; and 3) How to train goal-conditioned policies by offline reinforcement learning algorithms.

Generating High-Level Sub-Goals

While the low-level policy only requires a single sub-goal, we let the planner optimize over multiple sub-goals into the future to ensure an optimal long-term strategy. It can be formulated as a constrained optimization problem: maximizing the cumulative reward over sub-goals for multiple future steps with a constraint to force the sub-goals to be reachable by the low-level policy. The solution is a sequence of H sub-goals for H future high-level time steps. We select the first one to execute and repeat the planning process at the next time step.

Formally, The constrained optimization problem is formulated as follows:

$$\begin{aligned} &\max_{\mathbf{g}} V^{\pi}_{\text{env}}(\mathbf{s}_{t_0}, t_0) + \sum_{i=1}^{H} V^{\pi}_{\text{env}}(\mathbf{g}_{t_i}, t_i) \\ &\text{s.t.} \;\; V^{\pi}_{\text{TDM}}(\mathbf{s}_{t_0}, \mathbf{g}_{t_1}, t_0) + \sum_{i=1}^{H} V^{\pi}_{\text{TDM}}(\mathbf{g}_{t_i}, \mathbf{g}_{t_{i+1}}, t_i) \geq 0, \end{aligned} \tag{5.1}$$

where $\mathbf{g} = \left[\mathbf{g}_{t_1}^{\top}, \ldots, \mathbf{g}_{t_H}^{\top}\right]^{\top}$ is the sub-goal sequence of interest, $\mathbf{s}_{t_0}$ is the initial state, and

$$\begin{aligned} V^{\pi}_{\text{env}}(\mathbf{s}_{t_i}, t_i) &= \mathbb{E}\left[\sum_{j=1}^{N} r_{\text{env}}(\mathbf{s}_{t_{i,j}}, \mathbf{a}_{t_{i,j-1}}) | \pi\right], \\ V^{\pi}_{\text{TDM}}(\mathbf{s}_{t_i}, \mathbf{g}_{t_{i+1}}, t_i) &= \mathbb{E}\left[\sum_{j=1}^{N} r_{\text{TDM}}(\mathbf{s}_{t_{i,j}}, \mathbf{g}_{t_{i+1}}, t_{i,j}) | \pi\right], \end{aligned}$$

where r_{env} is the reward function defined in the original MDP environment, and r_{TDM} is the auxiliary goal-conditioned reward function. r_{TDM} is defined through a temporal difference model (TDM) that is commonly adopted in prior goal-conditioned policy learning works [98, 6, 36]. The TDM reward function is in the form:

$$r_{\text{TDM}}(\mathbf{s}_{\tau}, \mathbf{g}_t, \tau) = -\delta(\tau = t) d(\mathbf{s}_{\tau}, \mathbf{g}_t),$$

where $\delta(\cdot)$ is the indicator function, and d is a distance function specified in tasks. By definition, the TDM reward is always non-positive. The constraint in Problem (5.1) forces the expected cumulative reward r_{TDM} to be non-negative. Consequently, a feasible $\mathbf{g}$ has $r_{\text{TDM}} = 0$ for all the time steps, which means that $\mathbf{g}$ is guaranteed to be reachable.

The optimal solution of Problem (5.1) can be obtained by solving the following max-min optimization problem, where the objective is the Lagrangian function of Problem (5.1):

$$\max_{\mathbf{g}} \min_{\beta \geqslant 0} V^{\pi}(\mathbf{s}_{t_0}, \mathbf{g}_{t_1}, t_0) + \sum_{i=1}^{H} V^{\pi}(\mathbf{g}_{t_i}, \mathbf{g}_{t_{i+1}}, t_i), \tag{5.2}$$

where

$$\begin{aligned} V^{\pi}(\mathbf{s}_{t_i}, \mathbf{g}_{t_{i+1}}, t_i) &= V^{\pi}_{\text{env}}(\mathbf{s}_{t_i}, t_i) + \beta V^{\pi}_{\text{TDM}}(\mathbf{s}_{t_i}, \mathbf{g}_{t_{i+1}}, t_i) \\ &= \mathbb{E}\left[\sum_{j=1}^{N} r_{\text{env}} + \beta r_{\text{TDM}} | \pi\right]. \end{aligned} \tag{5.3}$$

The augmented value function V^{π} is essentially the value function of the low-level policy π regarding an augmented reward function r_g, which is the weighted sum of the original environmental reward and the auxiliary TDM reward:

$$r_g(\mathbf{s}_{\tau}, \mathbf{a}_{\tau}, \mathbf{g}_t, \tau) = r_{\text{env}}(\mathbf{s}_{\tau}, \mathbf{a}_{\tau}) + \beta r_{\text{TDM}}(\mathbf{s}_{\tau}, \mathbf{g}_t, \tau).$$

We may apply dual gradient descent to iteratively update g and the Lagrangian multiplier β. However, our hierarchical planning framework requires solving the optimization problem timely online. To this end, we consider the Lagrangian relaxation of Problem (5.2), which is obtained by specifying a prefixed value of β to relax the problem into an unconstrained maximization problem, where the multiplier β is essentially a trade-off factor between the objective and the regularization term:

$$\max_{\mathbf{g}} V^{\pi}(\mathbf{s}_{t_0}, \mathbf{g}_{t_1}, t_0) + \sum_{i=1}^{H} V^{\pi}(\mathbf{g}_{t_i}, \mathbf{g}_{t_{i+1}}, t_i). \tag{5.4}$$

We follow the convention to use the cross-entropy method (CEM) as the optimizer [32, 98] to solve Problem (5.4). In this chapter, we focus on tasks with high-dimensional images as observations. To this end, we need to sample meaningful images as valid goal states from the state space $\mathcal{S}$. In general, we do not have an explicit high-dimensional bound between meaningful images and invalid white noise, but it is possible to find a method that implicitly encourages valid samples. A natural method is modeling the goal distribution with a generative model such as VAE [98]. In particular, we adopt a CVAE [130] in this work.

The CVAE model consists of an inference mapping, i.e., an encoder $E_{\mu}(\mathbf{z}|\mathbf{g}_{t_i}, \mathbf{g}_{t_{i+1}})$, and a generative mapping, i.e., a decoder $D_{\nu}(\mathbf{g}_{t_{i+1}}|\mathbf{g}_{t_i}, \mathbf{z})$. The inference network maps a present goal state $\mathbf{g}_{t_i} \in \mathcal{S}$ to a latent state $\mathbf{z} \in \mathcal{Z}$ condition on its next goal $\mathbf{g}_{t_{i+1}}$, where $\mathcal{Z}$ is the latent space. The generative mapping, which conditions on the original input $\mathbf{g}_{t_i}$, maps the latent state $\mathbf{z} \in \mathcal{Z}$ to the next goal $\mathbf{g}_{t_{i+1}}$. The latent space $\mathcal{Z}$ is often low-dimensional, and the latent state $\mathbf{z}$ follows a specified prior distribution $p(\mathbf{z})$. By decoding goal states from latent variables sampled from the prior $p(\mathbf{z})$, we essentially sample goal states from a distribution approximating the goal distribution in the dataset. The sampled images are then more likely to be meaningful and correspond to in-distribution goal states. Different from the VAE used in [98], conditioning on the present state image reduces unnecessary variance. The sampled goals are those that are more likely given the current state. It improves sampling efficiency and leads to better optimization performance.

The objective function can then be optimized over the latent representation $\mathbf{z} = [\mathbf{z}_{t_1}^{\top}, \ldots, \mathbf{z}_{t_H}^{\top}]^{\top}$ of the goal images $\mathbf{g} = [\mathbf{g}_{t_1}^{\top}, \ldots, \mathbf{g}_{t_H}^{\top}]^{\top}$, where $\mathbf{g}_{t_{i+1}} = D_{\nu}(\mathbf{g}_{t_i}, \mathbf{z}_{t_{i+1}})$ is the reconstructed goal from sampled latent state $\mathbf{z}_{t_{i+1}}$ conditioned on the previous goal $\mathbf{g}_{t_i}$. An additional regularization term $-\lambda \log p(\mathbf{z})$ is added to the objective function to penalize $\mathbf{z}$ with low prior probability.

Goal-Conditioned Offline Reinforcement Learning

To train the low-level policy from static dataset, we aim to combine goal-conditioned training techniques with offline reinforcement learning. In particular, we need to train not only the policy π, but also the value function V^{π} in Eqn. (5.3).

Relabeling the Expert

The pre-collected datasets to train the goal-conditioned policies only include state and action transition pairs and corresponding rewards. There are no goals defined in training datasets originally, so we must create a goal for each sampled state-action pair during training. The trajectories are valid ones to reach any states within themselves, which makes it reasonable to use any state within each trajectory as its goal.

Therefore, following the hindsight experience replay [6], if we get a sampled pair ($\mathbf{s}^k_{t_{i,j}}$, $\mathbf{a}^k_{t_{i,j}}, \mathbf{s}^k_{t_{i,j+1}}, r^k_{\text{env},t_{i,j}}$) from the k-th trajectory in the training dataset, then we will relabel its goal as $\mathbf{g}^k_{t_{i+1}} = \mathbf{s}^k_\tau$ where $\mathbf{s}^k_\tau$ is a random future state within the whole trajectory. The new reward will be relabeled as:

$$r^k_{g,t_{i,j}} \leftarrow r^k_{\text{env},t_{i,j}} + \beta r^k_{\text{TDM},t_{i,j}}. \tag{5.5}$$

The TDM reward is computed regarding the labeled goal $\mathbf{g}^k_{t_{i+1}}$. The new pair ($\mathbf{s}^k_{t_{i,j}}$, $\mathbf{a}^k_{t_{i,j}}$, $\mathbf{s}^k_{t_{i,j+1}}$, $\mathbf{g}^k_{t_{i+1}}$, $r^k_{g,t_{i,j}}$) will be used as the training data to update the goal-conditioned value function and policy.

Robust to Distributional Shift

Many prior works on offline RL have dealt with the infamous distributional shift problems [74, 85]. Under the goal-conditioned setting, we need to additionally handle OOD goals. It is similar to the problem of OOD states. However, when dealing with OOD states, existing works require a dynamic model to estimate state uncertainty [85], which is extremely difficult to learn in high-dimensional observation space. Alternatively, we propose a new method to solve the distributional shift problem caused by OOD goals. Concretely, we perturb the goal in the sampled data point by a noise with a probability η, and then the reward $r^k_{g,t_{i,j}}$ is relabeled as the new noisy data point. In this way, we penalize the value at OOD goals, because a η-portion of the goals in the sampled data batch become noisy and were never achieved in the dataset. Thus, the corresponding TDM rewards will always be negative. Hence, the high-level goal planner using the same value function will be encouraged to avoid those low value areas at OOD goals, and thus it will implicitly affect the low-level policy during test time.

Practical Implementations

We now integrate the high-level goal planner and the low-level goal-conditioned policy to form a practical implementation, which we call the Hierarchical Goal-Conditioned offline reinforcement learning (HiGoC) framework. Given a static offline dataset containing K expert trajectories:

$$\mathcal{D} = \left\{ \{(\mathbf{s}^k_{t_{i,j}}, \mathbf{a}^k_{t_{i,j}}, \mathbf{s}^k_{t_{i,j+1}}, r^k_{\text{env},t_{i,j}})\}_{i,j}, k = 1, \dots, K \right\},$$

Algorithm 3: Training Procedure of Hierarchical Goal-Conditioned Offline Reinforcement Learning

```
1  Initialize: A Q-network $Q_\theta$ parametrized by $\theta$, A target network $Q_{\bar{\theta}} = Q_\theta$
     parametrized by $\bar{\theta}$, a policy network $\pi_\varphi$ parametrized by $\varphi$, an encoder $E_\mu$ and a
     decoder $D_\nu$ for the CVAE, a training dataset $\mathcal{D}$;
2  for step $c$ in range(0, $C$) do
3      Sample a batch of $b$ states $\mathbf{s}$ from the dataset $\mathcal{D}$;
4      Update $\mu$ and $\nu$ according to the CVAE objective;
5  end
6  for step $m$ in range(0, $M$) do
7      Sample a pair $(\mathbf{s}^k_{t_{i,j}}, \mathbf{a}^k_{t_{i,j}}, \mathbf{s}^k_{t_{i,j+1}}, r^k_{\text{env},t_{i,j}})$ ;
8      Sample a future state $\mathbf{s}^k_\tau$ within the $k$-th trajectory and add noise with a
         probability $\eta$ to obtain the goal $\mathbf{g}_{t_{i+1}} = D_\nu(E_\mu(\mathbf{s}^k_\tau) + \varepsilon, \mathbf{s}^k_{\tau-N})$;
9      Relabel the reward following Eqn. (5.5);
10     Update $Q_\theta$ with the CQL policy evaluation step and learning rate $\epsilon_\theta$:
         $\theta_m \leftarrow \theta_{m-1} + \epsilon_\theta \nabla_\theta J(\theta)$;
11     Update $\pi_\varphi$ according to the soft actor-critic style objective and learning rate $\epsilon_\varphi$:
           $\varphi_m \leftarrow \varphi_{m-1} + \epsilon_\varphi \mathbb{E}_{\mathbf{s}\sim d^{\pi_\beta}(\mathbf{s}),\mathbf{a}\sim\pi_\varphi(\mathbf{a})}[Q_\theta(\mathbf{s},\mathbf{a},\mathbf{g})$
               $- \log \pi_\varphi(\mathbf{a}|\mathbf{s},\mathbf{g})]$;
12     if $m$ mod target_update == 0 then
13         Soft Update the target network $\bar{\theta}_m \leftarrow (1-\tau)\bar{\theta}_{m-1} + \tau\theta_{m-1}$
14     end
15 end
```

the training procedure is shown in Alg. 3. A CVAE is first trained to model the goal distribution in the dataset. Then, the low-level goal-conditioned policy and the corresponding value function is trained through Conservative Q-Learning (CQL) [74], which is customized for the goal-conditioned settings. In the test phase, we follow Alg. 4 to query the hierarchical planner to control the agent.

5.5 Experiments

In this section, we present the results of our experiments with HiGoC. We intend to mainly answer the following questions with our experiments:

- Does HiGoC have better performance than the end-to-end RL policy and other hierarchical baselines?

Algorithm 4: Hierarchical Goal-Conditioned Planner

```
1 for high-level step $t_i$ in range(0, $T$) do
2     Solve the optimization problem in Eqn. (5.4) for a sub-goal $\mathbf{g}_{t_{i+1}}$;
3     for low-level step $t_{i,j}$, with $j$ in range(0, $N$) do
4         Sample $\mathbf{a}_{t_{i,j}} \sim \pi_\varphi(\mathbf{a} \mid \mathbf{s}_{t_{i,j}}, \mathbf{g}_{t_{i+1}})$;
5         Execute the action $\mathbf{a}_{t_{i,j}}$;
6     end
7 end
```

- Does longer look-ahead horizon H in the goal-selection optimization problem lead to better performance?
- Does the value function effectively encodes the optimality of different goal points?
- Do the perturbed goal sampling process and the CVAE improve the performance of the hierarchical planner?

We investigate those questions in simulated autonomous driving scenarios and robot navigation tasks. All experiments are evaluated and averaged over five random seeds and we report error bars corresponding to one standard deviation.

Experiment Settings

The CARLA Simulator

For autonomous driving scenarios, we evaluate our method in the CARLA simulator [44, 38, 22, 29], in particular, the driving tasks specified in [22]. In these tasks, the ego agent drive in a virtual urban town where the reward is given by the simulator evaluating the safety and efficiency of driving. There are 30 obstacle vehicles in total running in the whole map, so that we can simulate a realistic environment with intense interactions with surrounding vehicles. In our experiments, we first confine the evaluation in a local map containing a roundabout in the virtual town (Sec. 5.5). Afterwards, we extend the experiment to the whole town to evaluate the ability of our algorithm in building reliable policies in more general and realistic scenes (Sec. 5.5). We choose the "Town03" map from the CARLA simulator for evaluation.

In our experiments, static pre-collected training datasets are required to train the planner. To collect training data, an expert is trained using soft actor-critic (SAC) [51] method with the default reward function in each environment. We then execute the expert in the corresponding environment to record trajectories. After the collection of training datasets, we train HiGoC in the offline setting. During the test time, we drive the agent with the trained policies in the simulator.

We compose datasets with different levels of quality, namely "medium" and "expert" datasets. "Medium" datasets are collected by first training the expert using SAC, early stopping the training, and recording the trajectories with the half-trained expert in the environment. "Expert" datasets are collected with fully trained experts.

Antmaze in D4RL

In addition to the driving task, we evaluate our method on robot navigation tasks from offline RL benchmarks, which are the antmaze tasks in the D4RL dataset [44]. The dataset mainly consists of trajectories collected with undirected navigation of the simulated ant robot. The agent has to learn from the dataset to solve specific point-to-point navigation tasks. The task can be categorized into three difficulty levels: simple (i.e., umaze), medium, and difficult (i.e., large).

Variants and Baselines

We mainly compare HiGoC with two baselines: a) CQL [74], which is one of the state-of-the-art offline RL algorithms for end-to-end policy training; b) IRIS [92], which is also a hierarchical framework learned from offline dataset, with a high-level offline RL goal planner and a low-level imitation learning policy. For the antmaze experiments, we include another approach for comparison, OPAL [4], which is another hierarchical framework with a high-level strategy planner and a low-level primitive controller. Also, we compare different settings of HiGoC:

- **Different look-ahead horizon** H: Longer horizon enables the planner to better optimize long-term behavior. Meanwhile, errors of the value function and the CVAE model accumulate along the planning horizon. We are curious about how these two factors will trade off and affect the overall performance. We use the notation "H-step" to distinguish planners with different H. For instance, "2-step" refers to a planner with $H = 2$.

- **Different goal-sampling period** N: The goal-sampling period determines the episode length of the low-level goal-reaching MDP. As a result, it affects the accuracy of the learned value function. We specify the variants with different sampling period as "HiGoC-$N\Delta t$". For instance, "HiGoC-0.4s" refers to a variant with $N = 4$ since the sampling time of the simulator is $0.1s$.

- **Variant without goal perturbation**: The last variant we study is the one without goal sampling perturbations during training, denoted by "HiGoC-no noise". It has a goal-sampling time of 0.4s. By comparing it with the variant "HiGoC-0.4s", we would like to verify if the proposed method can effectively mitigate the issue of OOD goals and lead to better performance.

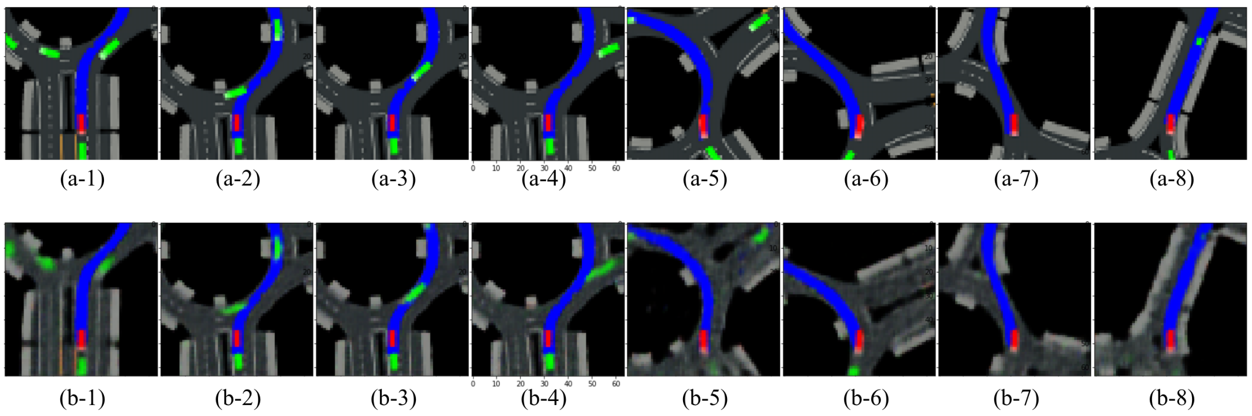


Figure 5.2: Visualization of the trajectory in the local map of a roundabout. We skip 10 frames in between these sampled images for brevity. (a) The actual observation images; (b) The corresponding goals selected by the goal planner with CVAE.

Evaluation Metric

The main evaluation metric is the normalized score (NS) [44]. The NS is defined as:

$$\texttt{NS = 100 * } \frac{\texttt{score - random score}}{\texttt{expert score - random score}}.$$

The score of each variant is the cumulative reward during test time after the whole training process finishes, namely, after 500K gradient steps in CARLA and 1M in antmaze. For the complex whole-town driving task, we also report the collision rate to give the audience a straightforward impression on the driving skill of the trained agents. We roll out 100 episodes, and compute the ratio of trajectories containing collisions.

Performance Comparison in CARLA

We first report the results of a comprehensive study on the small local map, comparing all the different settings listed in Sec. 5.5. The results are summarized in Table 5.1.

The Hierarchical Structure

We first compare HiGoC with the end-to-end policy baseline to evaluate the benefit of the hierarchical structure. From Table. 5.1, we can see that the performance of CQL is lower than all the other HiGoC variants (above 90.0). It is because the hierarchical planner is less greedy than the end-to-end policy. In Fig. 5.2, we show a sampled trajectory of the trained agent passing through a roundabout in the local map with the variant “7-step HiGoC-0.4s”. When the ego car in red is entering the roundabout, it first yields to the two green obstacle vehicles. Afterwards, it enters the roundabout and navigates through it successfully. When

Dataset	Look-ahead	HiGoC-0.4s	HiGoC-2.0s	HiGoC-no noise	IRIS	CQL
Medium	1-step	93.5 ± 9.4	98.4 ± 10.3	90.8 ± 6.9	83.3 ± 5.3	88.7 ± 6.3
	3-step	93.7 ± 10.1	95.5 ± 11.4	91.2 ± 9.8	87.2 ± 4.5	-
	5-step	96.2 ± 13.5	94.9 ± 10.2	95.5 ± 14.7	91.5 ± 3.1	-
	7-step	99.4 ± 12.9	92.7 ± 15.8	93.1 ± 13.3	90.1 ± 7.8	-
Expert	1-step	92.8 ± 6.5	97.4 ± 7.2	93.4 ± 6.9	84.1 ± 4.7	90.5 ± 4.1
	3-step	94.2 ± 6.8	96.5 ± 7.7	93.7 ± 7.3	92.3 ± 6.0	-
	5-step	97.9 ± 8.1	95.8 ± 9.4	94.6 ± 8.1	87.5 ± 5.2	-
	7-step	98.4 ± 9.4	91.6 ± 11.1	95.3 ± 10.5	89.9 ± 4.8	-

Table 5.1: Normalized scores of all variants when trained with the "Medium" and "Expert" quality dataset in the local map.

the ego vehicle is equipped with a non-hierarchical offline RL policy, it is very difficult for the ego car to slow down and yield to obstacles at the entrance of the roundabout. In contrast, even though the goals sampled by the CVAE are vague and twisted, they are still informative enough to guide the planner to be less aggressive and greedy.

Different Dataset Quality

We now compare the performance of HiGoC when trained on datasets with different levels of quality. When trained on the "Medium" dataset, HiGoC tends to achieve score that is closer to the corresponding expert. Since the data-collecting agent for the "Medium" dataset is controlled by an early-stopped policy, its policy is suboptimal with larger variance. Hence, the "Medium" dataset tends to cover larger state and action space. In contrast, the state and action distribution in the "Expert" dataset is much concentrated. The offline reinforcement learning agent is more suited in the settings where we have larger data coverage [117]. Therefore, it is reasonable that the agent is able to reach closer performance to the "Medium" level demonstration than the "Expert" level one.

Robust to Distributional Shift

In Table. 5.1, the normalized scores of "HiGoC-no noise" are consistently lower than "HiGoC-0.4s" in almost all the settings. It indicates that our method to increase the robustness against distributional shift is effective. It is worth noting that "HiGoC-no noise" still outperforms CQL on both datasets, demonstrating that the hierarchical structure is beneficial to solve the overall task.

Different Goal Planning Look-ahead

The overall planner is essentially a receding horizon controller, i.e., model predictive control (MPC). The optimization in Eqn. 5.4 is solved by CEM online during the test time, and the computing time is within 100ms which is sufficient for the framework to plan at a frequency of 10Hz. From control theory, if we have a precise dynamic model that can perfectly predict the behavior of the environment, the performance of the controller improves as the look-ahead horizon H increases [15]. In our high-level goal planner, the value function is equivalent to a prediction function estimating future cumulative rewards from the present state. Ideally, the trained agent should have better performance with longer H. However, we observed that the normalized score peaks at 7-step look-ahead with 0.4s goal-sampling period. The normalized score began to drop when further increasing H. If a goal-sampling period of 2.0s is selected, the highest normalized score is achieved with $H = 1$. It is because the learned value function becomes less accurate when predicting the far-away future. There is a trade-off between prediction accuracy and long-term planning capability.

Comparison with IRIS

We compare HiGoC against IRIS with different look-ahead horizons. Similar to ours, their high-level planner also selects a reference goal for the low-level goal-conditioned policy. The main difference is that IRIS chooses the optimal goal as the one with the highest expected future return in the original task MDP. In contrast, HiGoC chooses the sub-goal by finding the optimal sub-goal sequence over the look-ahead horizon, where the objective function is defined based on the value functions of the short-term goal-reaching MDPs between consecutive look-ahead time steps. Also, the low-level goal reaching policy of IRIS is learned with imitation learning, whereas ours is trained with offline RL. We ran IRIS with different look-ahead horizons, i.e., the number of timesteps between the current state and the goal point. As shown in Tab. 5.1, HiGoC consistently outperforms IRIS under different look-ahead horizons. Two factors contribute to this performance gain. Firstly, IRIS requires accurate estimation of the value function for the original task MDP, which is difficult for temporally extended tasks, especially under the offline setting. In contrast, HiGoC composes long-term behavior online based on the value functions of the short-term goal-reaching MDPs, which are easier to estimate via offline learning. Secondly, the low-level goal-reaching policy of IRIS is trained by imitation learning instead of offline RL. The offline RL agent is able to compose behavior that is better than the demonstration from the offline dataset.

A More Realistic Driving Scene

To further evaluate the ability of HiGoC in learning driving policies for more general and realistic scenes, we experiment HiGoC with a much larger driving scene in the CARLA simulator, the whole “Town03” map. We compare two settings with different look-ahead horizons, i.e., “7-step HiGoC-0.4s” and “1 step HiGoC-0.4s”. As shown in Tab. 5.2, both variants of HiGoC outperforms the baseline CQL, indicating the benefit of hierarchical structure in long-horizon tasks. In particular, HiGoC-7step has better performance. It further confirms that longer look-ahead steps can benefit the high-level strategy reasoning even in these complex tasks. Also, the performance of HiGoC is better than IRIS. It further confirms the advantage of HiGoC over IRIS in composing optimal long-term behavior. We notice that the agent reaches performance closer to the "Expert" than the "Medium" level demonstration in this complex scenario. One possible reason is that we use the same size of training dataset as the one in the local map, which is relatively small in this much larger scene. Although the "Medium" level dataset consists of more diverse samples, it does not have sufficient samples to cover high-reward state-action pairs in the whole town. Thus, it limits the training quality.

It is worth noting that the collision rate with our best policy is still too high for real-world application. We notice that most of the collisions are caused by the ego vehicle bumping into the rear-end of the preceding vehicle. It is because the current representation of observations only involves implicit velocity information (the historical positions of vehicles with faded color in the images) [21], and the lack of velocity information is also magnified by the encoding process of the CVAE. Nevertheless, we are still able to reduce the collision rate with the hierarchical architecture. It indicates that the hierarchy can prevent the agent from being too greedy.

Apart from the velocity information, it is also possible to further improve the performance from other aspects of representation. The current observation representation makes it necessary for the high-level planner to reason about the future road map in addition to the surrounding vehicles. In practice, it is a common practice for autonomous vehicles to have access to the map of the whole town in advance. If we leverage a representation with richer map information, it will no longer necessary for the high-level planner to forecast map change. Plus, a better representation will also enhance the capacity of the CVAE model. With a more accurate CVAE model, the sampled goal sequences are more likely to be realizable and optimal.

Dataset	**Variants**	**Normalized Score**	**Collision Rate**
Medium	HiGoC-7step	54.2 ± 9.7	0.32
	HiGoC-1step	45.5 ± 8.5	0.38
	IRIS-7step	43.3 ± 7.1	0.40
	CQL	31.1 ± 8.3	0.45
Expert	HiGoC-7step	61.6 ± 8.4	0.23
	HiGoC-1step	47.5 ± 7.9	0.32
	IRIS-7step	48.2 ± 8.7	0.33
	CQL	37.4 ± 6.2	0.36

Table 5.2: The performance in the whole town map.

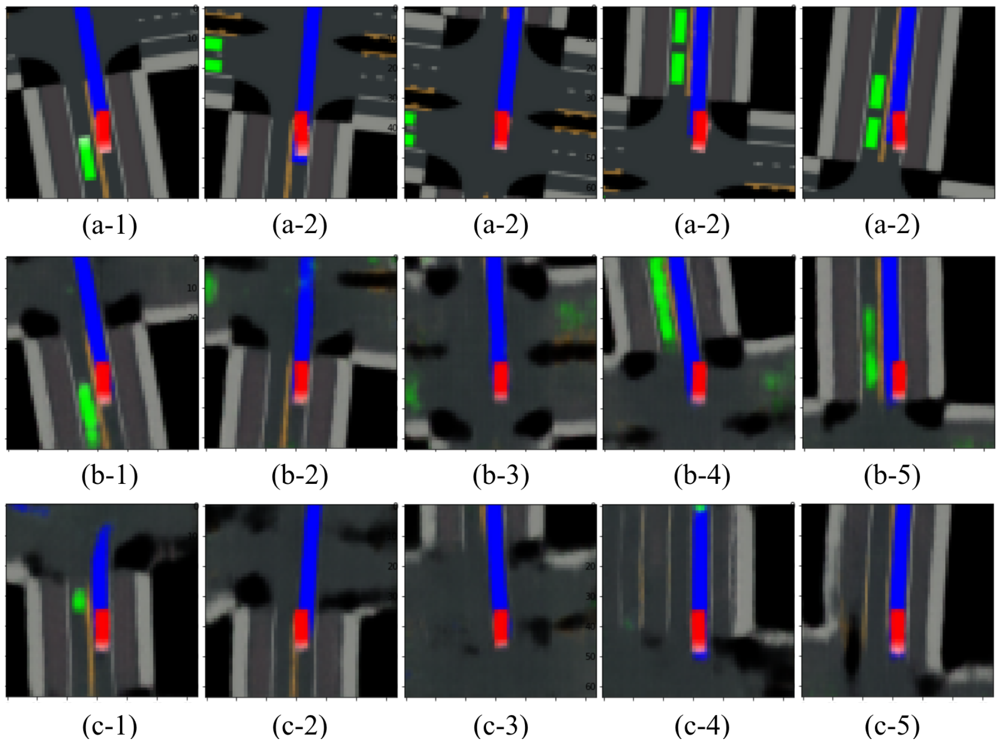


Figure 5.3: Visualization of a sampled trajectory. (a) The actual observation images; (b) The corresponding goals selected by the goal planner with CVAE; (c) The corresponding goals selected by the goal planner with VAE.

Performance in Antmaze

We now present the experimental results in the antmaze environments. As shown in Tab. 5.3, HiGoC has better performance than CQL and IRIS, especially in those challenging environments (i.e., medium and large). It is because the episode length increases with the size of the maze, which makes it more critical to compose long-term optimal behavior in order to succeed in these environments. The results further verifies the advantage of HiGoC over CQL and IRIS in temporally extended tasks. In Tab. III, we also report the scores of OPAL collected from their paper [4]. Since OPAL is deliberately designed to learn from offline data consisting of varied and undirected multi-task behavior, they only evaluated their methods on the medium/large-diverse antmaze environments. While HiGoC is not specially designed for diverse offline data, it still achieves performance close to OPAL in those environments.

Discussions

In this section, we discuss several remaining questions of interest with qualitative evaluations of the CVAE model and the learned value function.

Is CVAE better?

Whether the CVAE model can accurately model the goal distribution is important for the performance of the high-level planner. We are then curious about if we can indeed sample better goals from the CVAE model than a VAE. In Fig. 5.3, we compare the goal sequences sampled from the CVAE and VAE in the same scenario. From Fig. 5.3(c), we can see that the goals sampled from the VAE is vaguer than those sampled from CVAE in Fig. 5.3(b). More importantly, the vanilla VAE model does not correctly model the surrounding vehicles in the sampled goal sequences. As shown in Fig. 5.3(c-4) and (c-5), the obstacle vehicles in green are almost ignored by the vanilla VAE. In contrast, the CVAE model synthesizes the surrounding agents' future locations in its sampled goals, which allows the high-level planner to gain insights about the surrounding vehicles. As a result, the planner equipped with the CVAE reaches a higher score of 61.6 than 57.4 of that with the VAE in the whole town map when trained using the expert demonstration.

Is the Goal-Conditioned Value Function Reliable?

In Fig. 5.4(a), we plot the present observation when the red ego car is entering an intersection. The two images in Fig. 5.4(b) and (c) are two samples of goal candidates when starting from the present observation in Fig. 5.4(a). The green obstacle car should move down, but it should not be out of sight during the goal sampling period. Therefore, Fig. 5.4(b) with the green obstacle in sight is more reasonable than Fig. 5.4(c). Hence, Fig. 5.4(b) should be assigned a higher value than Fig. 5.4(c). Also, Fig. 5.4(c) is twisted and has plenty of noise. Compared with Fig. 5.4(b), it has lower probability in the latent space when it is encoded by the CVAE. A low probability means that the image is less likely to be a valid goal in the original image manifold. As shown in Fig. 5.4, the estimated value of Fig. 5.4(b) is 10.83 whereas 5.25 for Fig. 5.4(c). Hence, the learned value function can make reliable estimations on the quality of goal candidates.

Env	CQL	OPAL	IRIS	HiGoC
umaze	74.0	-	82.6 ± 4.7	85.3 ± 2.1
umaze-diverse	84.0	-	89.4 ± 2.4	91.2 ± 1.9
medium-play	61.2	-	73.1 ± 4.5	81.4 ± 2.4
medium-diverse	53.7	81.1 ± 3.1	64.8 ± 2.6	79.3 ± 2.5
large-play	15.8	-	57.9 ± 3.6	69.1 ± 2.3
large-diverse	14.9	70.3 ± 2.9	43.7 ± 1.3	67.3 ± 3.1

Table 5.3: Normalized scores on antmaze environments.

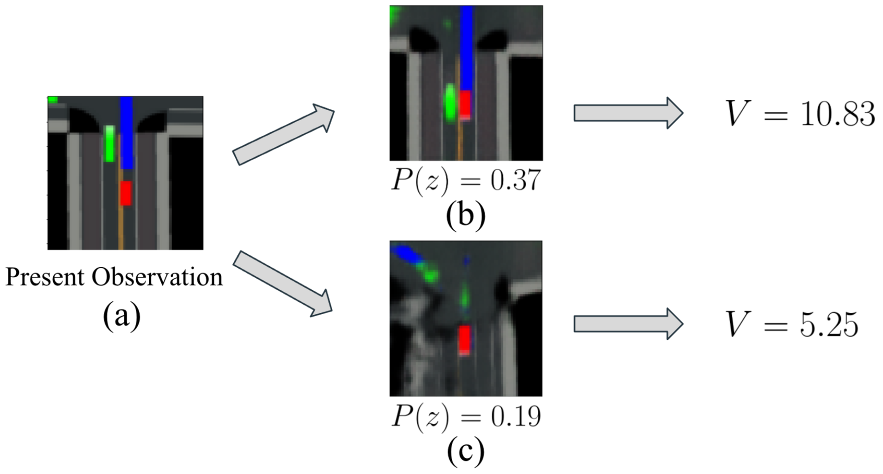


Figure 5.4: (a) The present observation image; (b) a high-quality goal candidate with high latent probability; (c) a low-quality goal candidate with low latent probability. The value estimation is higher for the high-quality candidate.

5.6 Conclusion

We propose a hierarchical planning framework through goal-conditioned offline reinforcement learning for tasks with extended temporal duration. The low-level policy is trained by offline RL in a goal-conditioned setting, which control the agent to achieve short-term subgoals. The offline training is improved by a perturbed goal sampling process to deal with distributional shift. The high-level goal planner takes advantage of model-based methods by composing behavior, and solves an optimization problem based on the low-level value function for long-term strategy. The proposed framework is empirically proved to be more suitable for temporally extended tasks than regular offline RL without hierarchy. The offline training strategy improves the robustness to distributional shift.

Part III

Looking Forward: Combining Offline and Online Training

Chapter 6

Guided Online Distillation: Promoting Safe Reinforcement Learning by Offline Demonstration

Safe Reinforcement Learning (RL) aims to find a policy that achieves high rewards while satisfying cost constraints. When learning from scratch, safe RL agents tend to be overly conservative, which impedes exploration and restrains the overall performance. In many realistic tasks, e.g. autonomous driving, large-scale expert demonstration data are available. We argue that extracting expert policy from offline data to guide online exploration is a promising solution to mitigate the conserveness issue. Large-capacity models, e.g. decision transformers (DT), have been proven to be competent in offline policy learning. However, data collected in real-world scenarios rarely contain dangerous cases (e.g., collisions), which makes it prohibitive for the policies to learn safety concepts. Besides, these bulk policy networks cannot meet the computation speed requirements at inference time on real-world tasks such as autonomous driving. To this end, we propose Guided Online Distillation (GOLD), an offline-to-online safe RL framework. GOLD distills an offline DT policy into a lightweight policy network through guided online safe RL training, which outperforms both the offline DT policy and online safe RL algorithms. Experiments in both benchmark safe RL tasks and real-world driving tasks based on the Waymo Open Motion Dataset (WOMD) [40] demonstrate that GOLD can successfully distill lightweight policies and solve decision-making problems in challenging safety-critical scenarios.

6.1 Introduction

Safe Reinforcement Learning (RL) aims to find a policy that not only achieves high rewards but also keeps the cost of violating constraints below a specified threshold. Traditional online safe RL algorithms [1, 88, 83] solve for an optimal safe policy by performing online rollouts in an environment and updating the policy accordingly. However, these algorithms always start training policies from scratch. The agent needs to learn to locate and avoid hazardous areas while it is still struggling to discover high rewards in the environment. The safety constraints discourage the agent from exploring certain hazardous areas [126], which leads to a pitfall that induces the policy to be overly conservative. The overly conservative policy often causes the agent to get stuck during its exploration, surrounded by complex hazard areas. Jammed at some states repetitively causes a skewed data distribution in the replay buffer, which deceives the policy that these states are the highest possible reward areas. It thus impedes the learning process and the overall performance.

In this situation, a near-optimal policy extracted from offline demonstrations can serve as a guide during online fine-tuning. Jump Start Reinforcement Learning (JSRL) [139], as an online fine-tuning method, has proven that training a new policy for online adaptation while using an offline extracted guide policy can be effective in regular RL settings, compared to naively initializing RL by the pre-trained policy [139]. It is natural and intuitive to propagate this meta-training scheme to the safe RL domain. The guide policy helps the agent being trained online start exploration from high-reward areas, and build new skills based on it thereafter. It is promising to save the agent from getting stuck in hazardous areas during exploration. Therefore, we propose to adapt JSRL to the safe RL setting, so that a better reward-cost trade-off can be achieved in those application scenarios where offline demonstration is available.

In many real-world situations, large-scale datasets already exist that can provide expert demonstrations for training policies[40, 23, 14, 17, 80, 148]. Prior work on imitation learning [59, 29] and offline RL [70, 86, 85] has investigated extracting high-performance policies directly from offline datasets to avoid risky online exploration or learning by trial and error. While it is seemingly promising to extract near-optimal policies with high rewards from offline datasets, prevalent Behavior Cloning (BC) or offline RL algorithms [94, 72] tend to fail when the demonstrations come from human experts. Decision transformer (DT) [24] has been shown as a strong method in such settings compared to these algorithms. It adopts large-scale models that are proven to have potentials [142, 13, 115]. Therefore, we explore the possibility of applying high-capacity decision transformers to learn from offline expert demonstrations in this chapter. However, easily accessible datasets often lack sufficient data points in safety-critical scenarios, such as collisions in real-world traffic datasets [35, 84]. Consequently, offline datasets alone cannot provide enough information on the safety constraints in the environment, and thus offline training is not sufficient for safe RL. It, therefore, strengthened the necessity of continuing to improve the decision-making policy by an online finetuning process with interactions in task environments [97].

Prior work [97, 158, 143, 133] typically uses the offline trained policy network architecture for online finetuning. Unfortunately, DT's transformer-based policy network, with its numerous parameters, can often fall short of meeting computation speed requirements in real-world tasks like autonomous driving. However, we do not intend to directly shrink the network size in offline training because it will sacrifice its performance to a great extent, and our experiments show that the performance and efficiency of online finetuning largely rely on the quality of the offline trained guide/expert policy. Alternatively, we sought to distill a more computationally efficient policy from DT during online training. There are many prior works on network distillation [119, 31, 136], but their student policies are trained either by supervised learning to replicate the teacher's behavior, which isolates the student policy and the environment forbidding active explorations or by indirect ways such as reusing the critic of the teacher, which does not fully incorporate the extracted prior skills from offline demonstrations. Training a different policy with the guidance of DT instead of initializing RL with existing policy in JSRL [139] allows us to change the policy network architecture and encourage the agent to explore more promising areas, which unifies the purpose of finetuning and distillation in this chapter.

In summary, we propose a training scheme, named Guided Online Distillation (GOLD), for safe RL tasks where offline expert demonstration is available. The details can be found in [82]. GOLD leverages an offline learned large-scale policy to guide the online learning of a computationally efficient, safe RL policy. Compared to safe RL from scratch, GOLD can improve the cumulative reward achieved by the policy while maintaining the cumulative cost below the threshold. In summary, our contributions include:

- We propose a training scheme, Guided Online Distillation (GOLD), for safety-critical scenarios where offline expert demonstration is available. It solves the problem caused by limited high-risk cases in offline datasets and conservative exploration in safe RL.
- We empirically show that adopting a DT instead of BC improves the performance of the offline extracted policy, and the large-capacity and well-performed DT guide policy is crucial for the online distilled lightweight policy to optimize its reward-cost trade-off.
- We train and evaluate the proposed algorithm on both benchmark safe reinforcement learning and real-world autonomous driving tasks extracted from the Waymo Open Motion Dataset (WOMD) [40, 23]. We show that GOLD can effectively accelerate online learning and improve policy performance.

6.2 Preliminaries

Constrained Markov Decision Process

We define a Constrained Markov Decision Process (CMDP) by a tuple $\mathcal{M} := (\mathcal{S},\ \mathcal{A},\ \mathcal{P},\ r,\ c,\ \gamma,\ \mu_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function specifying the probability $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$ from state $\boldsymbol{s}_t$ to $\boldsymbol{s}_{t+1}$ when applying $\boldsymbol{a}_t$, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $c : \mathcal{S} \times \mathcal{A} \to [0, C_m]$ is the cost function for violating the constraint with C_m as the maximum cost [5], γ is the discount factor, and $\mu_0 : \mathcal{S} \to [0, 1]$ is the initial state distribution.

In safe RL, the goal is to find a policy $\pi \in \Pi$ where Π is the policy class such that it obtains a high return in reward and maintains the cost return below a threshold $\kappa \in \mathbb{R}^+$. Formally, we denote the reward value function $V_r^\pi(\mu_0) = \mathbb{E}_{\tau\sim\pi,\boldsymbol{s}_0\in\mu_0}[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t)]$ as the

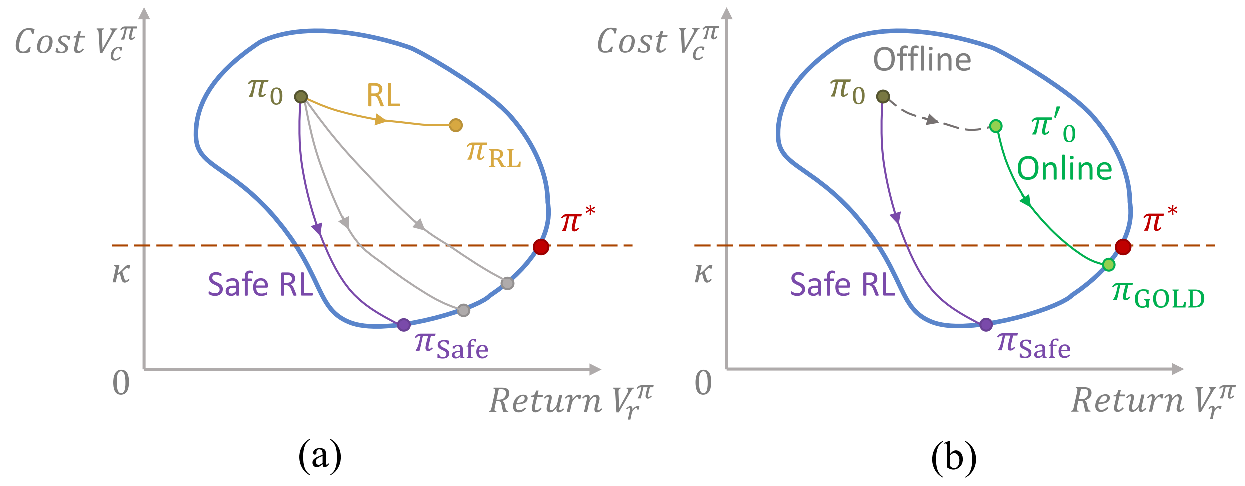


Figure 6.1: An illustration of reward-cost relation for all policies in a certain environment. (a) Regular and safe RL policies converge to points far away from the optimal policy π^*. (b) Trained from offline demonstration, π'_0 is attracted toward π^*, and hence our method results in a lightweight yet more capable policy during online distillation.

discounted cumulative reward under the policy π and the initial state distribution μ_0, where $\tau = \{\boldsymbol{s}_0, \boldsymbol{a}_0, \dots\}$ is the trajectory. The cost value function is defined similarly as $V_c^\pi(\mu_0) = \mathbb{E}_{\tau\sim\pi,\boldsymbol{s}_0\in\mu_0}[\sum_{t=0}^{\infty}\gamma^t c(\boldsymbol{s}_t, \boldsymbol{a}_t)]$. The objective is then to find an optimal policy π^* by solving the following constrained optimization problem:

$$\pi^* = \arg\max_{\pi} V_r^\pi(\mu_0), \text{ s.t. } V_c^\pi(\mu_0) \leq \kappa. \tag{6.1}$$

Reward-Cost Relationship

The constraint in Eqn. 6.1 illustrates that a safe RL algorithm needs to control two signals simultaneously, i.e., reward and cost, compared to regular RL. A plot of cumulative cost and reward pair of policies on a 2d-plane is informative for safe RL algorithm analysis. Given a specific environment, each policy $\pi \in \Pi$ can be mapped onto a point within the blue circle, representing the reward-cost return pair (V_r^π, V_c^π) that π obtains in the environment [89]. All policies that lie below the threshold κ (the orange dash in Fig. 6.1) are considered feasible solutions. The optimal policy π^* obtains the highest possible reward return while maintaining the cost return below the threshold κ. When a policy is being trained in the environment, its corresponding reward-cost pair moves on this 2d-plane. Most RL algorithms randomly initialize a policy π_0, which places it on the upper left corner in Fig. 6.1. If being evaluated, π_0 will obtain low reward and high cost.

Assume RL algorithms are effective in the environment. For a regular RL algorithm, e.g., PPO [124] or SAC[50], this means the reward obtained by the policy continuously increases, but there is no consideration of the cost return. This results in the yellow trajectory in Fig. 6.1(a). On the contrary, a safe RL algorithm [76] is dedicated to decreasing the cost and increasing reward simultaneously. Hence, its trajectory moves toward the bottom right corner in Fig. 6.1(a).

Once a trajectory reaches the boundary of the feasible area below κ, it stops moving because the safe RL does not allow an increase in cost or a decrease in reward. Therefore, if a safe RL algorithm penalizes too hard on cost return V_c^π, its trajectory will end up at a point on the boundary that has a low reward. The fast drop in cost during training usually leads to a convergence point π_{Safe} that is far away from the optimal policy π^*. This aligns with our observation in experiments where safe RL agents often get stuck in a position surrounded by hazards and cannot find a way out.

In this case, offline policy extraction from expert demonstrations can provide a head start, which boosts the original initial π_0 to π_0' (closer to π^*) in Fig. 6.1(b). The online distillation can start from π_0' that skips exploring the environment from scratch and thus requires less effort to π_{GOLD} of higher quality than π_{Safe} even with the same online training RL backbone. Thus, we propose a new training scheme that pushes the trajectory toward the optimal policy π^* by leveraging demonstrations to extract prior skills and perform online safe RL finetuning.

6.3 Guided Online Distillation

In this section, we present our proposed method: Guided Online Distillation (GOLD). It consists of two stages: 1) extracting a large-scale guide policy from offline demonstration, and 2) distilling a robust but lightweight policy through online exploration with the guidance of the guide policy.

Extracting Expert from Offline Demonstration

Offline policy training from demonstration has been a popular research topic, and many methods have been proposed. DT [24] is an approach that lies in between BC and offline RL and proves to be competent. It adopts a similar loss function and training scheme as BC but also considers reward signals as offline RL. We therefore choose to apply DT to extract expert policies from offline demonstration and empirically show that it is superior to both BC and offline RL for safety-critical navigation and autonomous driving tasks.

The trajectory representation and model architecture follow the design in [24]. Specifically, we choose to represent the trajectory by three modalities: observation, action, and returns-to-go. Formally, the trajectory representation is $\tau = \left(\hat{R}_1, \boldsymbol{s}_1, \boldsymbol{a}_1, \ldots, \hat{R}_T, \boldsymbol{s}_T, \boldsymbol{a}_T\right)$, where $\hat{R}_t = \sum_{i=t}^{T} r_i$ is the returns-to-go, $\boldsymbol{s}_t$ and $\boldsymbol{a}_t$ are the observation and the action at time t. The model is fed with the most recent K timesteps, encompassing a total of $3K$ tokens. In the experiments in this chapter, we find the default setting of $K = 20$ to be suitable for most of the tasks. A GPT [113] model processes the inputs by autoregressive modeling.

Leveraging a dataset of offline trajectories, we extract minibatches with a sequence length of K from the dataset. The prediction head linked to the input token o_t is trained to predict action a_t. The loss is only evaluated on the predicted action, as no performance gain is reported by predicting observation and returns-to-go [24]. In our case, the loss is defined to be

$$L_{DT} = ||\boldsymbol{a} - \hat{\boldsymbol{a}}||^2 \tag{6.2}$$

where $\boldsymbol{a}$ is the ground truth action, and $\hat{\boldsymbol{a}}$ is the predicted action by the DT.

Online Policy Distillation

DT for expert policy extracting from offline demonstration improves the performance, but it sacrifices computation efficiency and robustness. Transformers are bulk in size, so they consume large amount of computation resources. This can be critical when deploying them as decision-making modules on real systems that request fast response frequency. The offline demonstration is also not comprehensive, and hence there are always hazardous corner cases not included. It results in a guide policy that is only reliable close to the in-distribution areas within the offline data support. Therefore, we propose to distill a lightweight policy network and improve its robustness to out-of-distribution hazardous areas with the guidance of DT during online exploration within the task environment.

The backbone of online distillation is based on JSRL [139]. We define a guide policy as the pre-trained DT from Sec. 6.3, whose parameters are frozen during online distillation. The policy network to be distilled is designed to be a lightweight Multi-Layer Perceptron (MLP). On the one hand, JSRL makes sure the reward maintains its stable improvements, instead of dropping dramatically in naive online fine-tuning methods. The skills of the guide DT are distilled into the lightweight policy by exposing it to a high-reward trajectory distribution. On the other hand, the states induced by the guide policy are also relatively safe, where the agent explores to learn fine-grained information on the costs. This makes the lightweight exploration policy not only training efficient but also robust.

However, storing the rollouts of both guide and lightweight policies in one replay buffer causes a mixed data distribution, which induces problems for the critic learning in RL algorithms. Actor-critic methods aim to approximate an optimal Q-function corresponding to the current parameterized policy $\pi(\boldsymbol{a}|\boldsymbol{s})$, which satisfies the equation

$$Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \\ \gamma \mathbb{E}_{\boldsymbol{s}_{t+1} \sim T(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t), \boldsymbol{a}_{t+1} \sim \pi(\boldsymbol{a}_{t+1}|\boldsymbol{s}_{t+1})} \left[Q^{\pi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) \right].$$

The Q-function should be evaluating the future cumulative reward under the data distribution induced by the current policy. If the training data is collected by multiple policies as the online distillation process, the mixed and skewed data distribution will cause the Q value prediction to be inaccurate on trajectories collected by the current lightweight policy being trained.

We propose to resolve the aforementioned problem by leveraging Implicit Q Learning (IQL) [70] as the training algorithm during online distillation. IQL approximates a Q-function without an explicit policy by expectile regression. Specifically, it first estimates expectiles only with respect to the actions in the support of the data by first approximating a value function $V_\psi(\boldsymbol{s})$ with a loss $L_V(\psi)$,

$$L_V(\psi) = \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim\mathcal{D}}\left[L_2^\tau(Q_{\hat{\theta}}(\boldsymbol{s},\boldsymbol{a}) - V_\psi(\boldsymbol{s}))\right],$$

where $L_2^\tau(u) = |\tau - \mathbb{1}(u<0)|u^2$, and τ is the expectile. It then avoids injecting stochasticity from the state distribution by averaging over the stochasticity from the dynamics transitions and fitting a Q-function Q_θ with a loss $L_Q(\theta)$,

$$L_Q(\theta) = \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}')\sim\mathcal{D}}\left[r(\boldsymbol{s},\boldsymbol{a}) + \gamma V_\psi(\boldsymbol{s}') - Q_\theta(\boldsymbol{s},\boldsymbol{a})\right]^2.$$

The fitted Q-function corresponds to the upper expectile of the returns, which makes it approximate better the Q-function corresponding to the optimal policy. This decoupling between the Q-function approximation and the current policy is suitable for GOLD. The Q-function is not sensitive to the mixed state trajectory distribution in the replay buffer anymore, instead, it corresponds to the optimal policy.[148]

Practical Implementation

We summarize the complete proposed algorithm GOLD in Algo. 5. A DT is first trained from offline demonstration, which later serves as the guide policy during online distillation. A lightweight exploration policy network is then trained interactively in the task environment by IQL. In GOLD, the safety constraints are enforced by reward shaping, adapting IQL to safe RL settings, i.e., its actual reward is a linear combination of the original reward and cost

$$\text{reward}_{\text{new}} = \text{reward} + \lambda \cdot \text{cost},$$

since we are focused on safety-critical tasks in this chapter.

6.4 Experiments

Experiment Setting

Safety Gym & Bullet Safety Gym

`safety-gym` [118] and `bullet-safety-gym` [47] are open-source frameworks which is designed to train and evaluate safety performance across many tasks and environments, distinct in complexity and design. The observation of an agent is set to include the agent's own body state, the sensing information on obstacles given by pseudo laser rays, and task-specific information such as distance to goals. We pick five tasks with two different agent types. The tasks include `Circle` and `Gather`, `Goal`, `Button`, `Push`, and the agent types are `Point` and `Car`. We perform training and evaluation in different combinations of tasks and agents.

Algorithm 5: Training Procedure of GOLD

```
1  Initialize: A decision transformer (DT) $\pi^g_\mu$ for guide policy, an lightweight policy
     network $\pi_\varphi$, a Q-network $Q_\theta$, A target network $Q_{\bar{\theta}} = Q_\theta$, a training dataset $\mathcal{D}$, a
     replay buffer $\mathcal{B}$;
2  // Prior skills extraction from offline demonstration
3  for step n in range(0, N) do
4      Sample a batch of $b$ trajectory segments $\tau^t_{t-H}$ from the dataset $\mathcal{D}$;
5      Update $\mu$: $\mu_n \leftarrow \mu_{n-1} + \epsilon_\mu \nabla_\mu L_{DT}$
6  end
7  // Online distillation procedure
8  for guide step h in $[H_1, H_2, \dots, H_m]$ do
9      Assign a non-stationary policy $\pi$ defined at each timestep: $\pi_{1:h} = \pi^g_\mu$,
         $\pi_{h+1:H} = \pi_\varphi$;
10     Collect rollouts by $\pi$ and append them to the replay buffer $\mathcal{B}$;
11     for train step m in range(0, M) do
12         Sample a batch $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$ from $\mathcal{B}$;
13         Update $Q_\theta$ and $\pi_\varphi$ by IQL;
14     end
15 end
```

MetaDrive

a lightweight yet powerful driving simulator [87], which provides convenient scene composition with various road maps and traffic settings that are critical for generalizable RL. The simulation is realistic as it leverages an accurate physical engine and emulates sensory input. The driving scenes can be replayed from real-world traffic data such as WOMD [40, 23], NuScenes [14], Argoverse [17], etc. The observation consists of pseudo Lidar-like cloud points, navigation information represented by waypoints, and ego states, including steering, heading, velocity, and relative distance to boundaries. The action space contains the acceleration and steering of the ego vehicle.

Offline Datasets

In our problem setting, we assume access to offline datasets collected from expert demonstration. These demonstrations are near-optimal, which contain few safety-critical situations, but mostly trajectories with high reward returns. Formally, the dataset contain N expert trajectories, each of which is represented by H tuples $\{(\boldsymbol{s}^k_t, \boldsymbol{a}^k_t, \boldsymbol{s}^k_{t+1}, r^k_t)^T_{t=0}\}^N_{k=1}$, where $t \in [0, T]$ is the time step from 0 to H, and $k \in [1, N]$ is the episode number from 1 to N. In the `(bullet)-safety-gym` environments, the offline datasets are collected by expert RL policies to imitate human experts, which are trained in online settings by SAC with carefully tuned reward shaping for data collection purposes. These expert RL policies are able to reach high

		Offline		Online				
Task		**BC**	**DT**	**IQL**	**CVPO**	**GOLD (BC-IQL)**	**GOLD (DT-CVPO)**	**GOLD (DT-IQL)**
Car-Circle	r $\uparrow$	366.9 ± 10.4	**450.3 ± 53.8**	630.7 ± 26.4	502.5 ± 10.8	628.4 ± 20.6	613.7 ± 26.3	**688.3 ± 4.2**
	c $\downarrow$	41.4 ± 5.3	**40.4 ± 6.3**	17.5 ± 2.8	7.4 ± 3.3	13.6 ± 1.4	3.9 ± 0.5	**3.2 ± 1.5**
Car-Gather	r $\uparrow$	5.6 ± 2.31	**7.1 ± 1.52**	10.3 ± 1.22	10.2 ± 0.87	12.8 ± 2.63	11.9 ± 0.44	**14.0 ± 1.98**
	c $\downarrow$	0.42 ± 0.17	**0.38 ± 0.16**	0.23 ± 0.12	0.18 ± 0.04	0.19 ± 0.27	0.15 ± 0.02	**0.14 ± 0.04**
Point-Goal	r $\uparrow$	19.2 ± 1.4	**24.1 ± 0.5**	31.6 ± 3.2	32.1 ± 5.3	32.1 ± 5.8	31.4 ± 3.8	**33.9 ± 6.5**
	c $\downarrow$	16.7 ± 2.4	**15.2 ± 4.6**	10.5 ± 3.9	8.5 ± 1.4	**8.0 ± 3.8**	11.5 ± 2.9	8.3 ± 1.3
Point-Button	r $\uparrow$	23.7 ± 5.2	**27.8 ± 4.2**	35.1 ± 4.7	38.2 ± 2.4	41.1 ± 3.1	39.2 ± 2.4	**44.8 ± 3.1**
	c $\downarrow$	18.5 ± 5.9	**17.3 ± 2.5**	8.4 ± 2.1	7.5 ± 2.8	**6.2 ± 4.2**	6.8 ± 2.9	6.5 ± 1.1
Point-Push	r $\uparrow$	2.1 ± 3.2	**4.6 ± 2.1**	5.3 ± 1.0	2.5 ± 0.3	6.6 ± 1.6	4.2 ± 1.1	**8.0 ± 2.3**
	c $\downarrow$	50.2 ± 8.1	**45.4 ± 5.8**	34.1 ± 9.1	19.3 ± 4.2	24.3 ± 3.8	**18.3 ± 4.5**	20.4 ± 6.9
Car-Goal	r $\uparrow$	13.2 ± 1.7	**16.4 ± 3.4**	22.8 ± 1.3	19.8 ± 1.9	27.9 ± 2.1	28.4 ± 1.2	**30.5 ± 5.4**
	c $\downarrow$	53.4 ± 2.1	**49.9 ± 6.3**	20.4 ± 5.2	24.3 ± 6.6	14.6 ± 2.6	12.2 ± 4.1	**10.8 ± 3.2**
Car-Button	r $\uparrow$	19.6 ± 2.2	**26.5 ± 3.7**	28.9 ± 10.4	27.1 ± 3.8	36.1 ± 3.0	30.5 ± 3.8	**42.0 ± 2.6**
	c $\downarrow$	34.1 ± 10.5	**20.8 ± 9.3**	12.5 ± 6.8	8.8 ± 1.3	9.8 ± 5.0	7.1 ± 4.3	**6.5 ± 6.1**
Car-Push	r $\uparrow$	1.5 ± 0.1	**2.6 ± 0.2**	3.8 ± 1.1	3.0 ± 0.4	4.5 ± 0.2	4.2 ± 0.2	**5.1 ± 0.4**
	c $\downarrow$	65.3 ± 9.8	**58.9 ± 11.6**	23.3 ± 2.9	19.3 ± 5.8	19.4 ± 1.9	**15.7 ± 2.3**	17.4 ± 1.1
MetaDrive Waymo	r $\uparrow$	115.78 ± 132.89	**133.48 ± 190.50**	26.29 ± 68.82	113.48 ± 163.84	141.93 ± 189.54	115.66 ± 163.18	**143.69 ± 175.98**
	c $\downarrow$	**1.14 ± 1.96**	1.25 ± 1.92	2.57 ± 2.36	1.05 ± 1.84	**1.03 ± 1.85**	1.25 ± 2.07	1.15 ± 2.05
	sr $\uparrow$	53%	**54%**	40%	58%	63%	62%	**73%**

Table 6.1: The performance of offline policy extraction and online policy distillation. Metric notations are defined as, r: reward, c: cost. For the realistic driving environment based on WOMD, we also compare success rate as sr.

reward returns and satisfy cost requirements. We use Waymo Open Datasets as the offline demonstration dataset for the MetaDrive task. These datasets are recordings of real traffic scenes, which are generated by human drivers.

Baselines

We compare our proposed GOLD to its own variants and state-of-the-art safe RL methods, including:

- **Safe RL Trained from Scratch**: To show the role of the guide policy in GOLD, we choose safe RL methods: Implicit Q Learning (IQL) [70] equipped with reward shaping and Constrained Variational Policy Optimization (CVPO) [88] as baselines, which are both trained from scratch. They are not warm-started by the guide policy (DT), since we intend to keep the number of parameters and network structure the same as GOLD.
- **Variants of the proposed method**: We also demonstrate how different components in the proposed method contribute to the final performance. For the guide policy trained from offline datasets, we compare DT with BC. For the RL backbone of online distillation and training, we compare IQL with CVPO. In summary, we have four variants, namely, GOLD (BC-IQL), GOLD (DT-CVPO), and GOLD (DT-IQL).

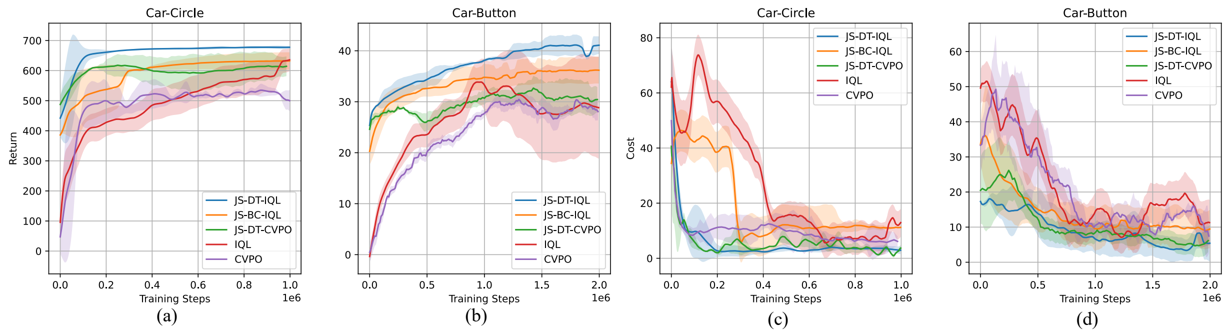


Figure 6.2: The learning curves of GOLD (DT-IQL) and baselines. (a)(c): The reward and cost curves in Car-Circle. (b)(d): The reward and cost curves in Car-Button.

Guide Policy Performance

The performance of various offline policy extraction methods in terms of both reward and cost is listed in Tab. 6.1. The columns of offline methods show that DTs are superior in both reward and cost than BC in all benchmark tasks. In convention, BC or RL adopts MLP as the policy or value network. However, DTs use large models such as pre-trained GPT2 as the network backbone. This difference dramatically increases the model capacity of the policy network and thus improves the final performance by a large margin.

We observe the performance of DT is correlated with the offline dataset size. Typically, it benefits from enlarging the size of the dataset. We find it is sufficient to show the difference between DT and MLP using datasets of a scale of 100k trajectories for `(bullet-)safety-gym` tasks, and 10k trajectories from WOMD for Metadrive tasks. Due to limited hardware accelerator resources, we cannot perform more computation-intensive guide policy extraction on larger datasets. We leave it to future work as it pertains to the current trend of leveraging richer and bulkier datasets.

Online Distillation Evaluation

With the guide policy extracted from offline demonstrations, our proposed method finetunes and distills a lightweight yet more powerful policy network through interactions within online environments.

Computation Efficiency

The online training distills a much smaller policy network, i.e. a two-layer MLP with a hidden size of 256, which is standard in most RL problem settings. The number of parameters of the MLP is negligible compared to the huge transformer used by the guide policy, which usually has 10x times more parameters (670k in `safety-gym` tasks). The computation efficiency is thus noticeable and becomes an advantage when deploying these MLP policies compared to the huge DT, if the performance is above threshold. For the benchmark tasks in our setting, the online distilled MLP runs at 0.03s per 100 inference runs, compared to 0.31s per 100 inference runs of DT on a single NVIDIA GeForce RTX 2080 Ti GPU.

Policy Performance

The performance of all baselines and variants are listed in Tab. 6.1 under the tab "Online". Our proposed method outperforms all baselines in terms of reward and is superior in most tasks in terms of cost. We can see that algorithms with a guide in online distillation, i.e. variants of GOLD, perform better than training from scratch, i.e., IQL and CVPO. Regular safe RL algorithms tend to get stuck with local optimal solutions because they are discouraged from high-risk exploration in the environment by their cost constraints. However, with the guidance of pre-trained offline policies, GOLD avoids the exploration that leads to many failures before success and can discover highly lucrative solutions. Fig. 6.2 shows that the agent learns faster and better when equipped with a guide policy.

We also show the advantage of guidance during online distillation in Fig. 6.3. Here, the red car intends to press the yellow button on the upper right corner starting from the lower left corner, without touching the purple and blue hazardous obstacles. The red curve behind the car is its historical trajectory. The purple obstacles are moving, while the blue obstacles are staying still. In Fig. 6.3(a), the ego car trained with CVPO only finds a local optimal solution and chooses to avoid the bottom purple box at the beginning, which results in zigzagging trajectory and hence lower reward in the later stage of the episode. In contrast, the ego car trained with our method learns to find the most direct and safe way to the goal by the guidance of the expert policy in the early training stage and thus obtains higher reward and lower cost.

In Tab. 6.1, we also show that the online distillation performance improves with a better guide policy. The algorithms GOLD (DT-*) usually obtain better rewards compared to GOLD (BC-IQL). This aligns with the intuition that a better teacher reduces the effort to learn the same level of skills.

Plus, GOLD (DT-IQL) typically performs better than GOLD (DT-CVPO), as shown in Tab. 6.1 and Fig. 6.2, even though they both adopt DT as the guide policy. This is because of the data distribution mismatch in the replay buffer, which is mentioned in Sec. 6.3. CVPO learns Q-functions evaluating only the current explore policy being trained, which mismatches the trajectories collected by a mixture of guide and explore policies. Using IQL as the online finetuning and distillation backbone solves this problem because it learns Q-functions only by evaluating the optimal policy in the environment, which in theory, can learn from data collected by any policy.

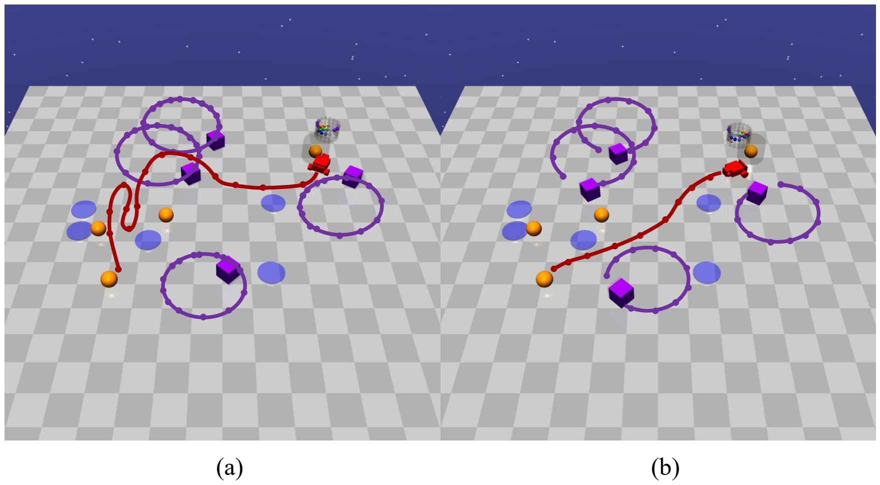


Figure 6.3: Sampled trajectories in the Car-Button task. (a): GOLD (BC-IQL). The red ego car avoids the moving purple hazardous obstacle at first and struggles to find its way in the later stage. (b): GOLD (DT-IQL). The ego car can find an efficient trajectory to reach the goal, thanks to the prior knowledge inherited in the guide DT policy.

Realistic Experiments in Driving Scenarios

Our method is applicable to and effective in realistic scenarios, which we demonstrate by experiments on MetaDrive. These experiments are fairly close to real-world scenes because we make MetaDrive replay vehicle trajectories from WOMD. The observations input to the ego agent, including Lidar cloud points, navigation information, and ego states, also resemble the real-world setting. The goal of the ego vehicle is to arrive at a specific target position defined in WOMD. We randomly choose 10k scenarios from WOMD for training and 1k scenes for testing.

As shown in Tab. 6.1, our method surpasses baselines by around 15% in reward and maintains the cost below the threshold. The success rate is increased by 12% compared to the best variant, which shows GOLD is capable in realistic driving tasks. The evaluation results and analysis on previous benchmarks in Sec. 6.4 are transferrable to realistic tasks, which confirms the performance of GOLD is correlated to the guide policy quality. This supports and justifies our design of offline policy extraction by DT.

The driving experiments further validate that bringing in prior skills during online distillation is necessary for learning high-quality policy in real-world safety-critical scenarios. CVPO or IQL from scratch is too conservative to explore because it is almost impossible to discover useful skills without severe cost violations. GOLD skips the risky exploration in this safety-critical environment. With the offline trained DT as guidance, GOLD can distill and improve a lightweight policy network without struggling to jump out of the most hazardous areas. Also, CVPO outperformed IQL by a large margin when trained from scratch, but GOLD (DT-IQL) surpasses GOLD (DT-CVPO). This confirms that IQL's decoupling of Q-function and policy training works seamlessly with GOLD. More video demonstrations can be found on `https://sites.google.com/view/guided-online-distillation`.

6.5 Conclusion

We propose a new offline-to-online training scheme named Guided Online Distillation for safety-critical tasks. A large-scale guide policy is first extracted from offline demonstrations. It serves as a guide for online distillation, where a lightweight policy is distilled through interactions with the task environment. This lightweight network can meet computation speed requirements in realistic settings, in contrast to the bulk guide policy. The guided distillation saves the policy from being repeatedly exposed to hazards during its exploration to find useful skills, which improves its training efficiency and final performance. Experiments in both benchmarks and real-world driving experiments based on the WOMD show that the distilled policy by GOLD surpasses safe RL baselines that are trained from scratch.

Chapter 7

Final Words

7.1 Summary of Key Contributions

This dissertation has made significant strides in advancing the field of safe and trustworthy decision-making through reinforcement learning (RL) in real-world scenarios, particularly focusing on autonomous vehicles (AVs) and their interactions in dynamic environments. The core contributions of this work lie in the development and validation of innovative RL frameworks and algorithms, each addressing specific challenges in the realm of autonomous driving.

Chapter 2 presented a Partially Observable Markov Decision Process (POMDP) framework for behavior planning in lane-changing scenarios, emphasizing the interactions between autonomous cars and human drivers. This approach not only accounted for the variability in human behavior but also enhanced the predictability and safety of AVs in such scenarios. The use of real-world data for training and validation sets a new benchmark in the field, moving beyond simulations to practical, real-world applications.

Chapter 3 introduced the Pessimistic Offline Reinforcement Learning (PessORL) framework, tackling the challenge of out-of-distribution states in offline RL. This project's significance lies in its theoretical foundations and practical implications, demonstrating improved performance in handling OOD states, a crucial factor for the safe deployment of RL agents in real-world settings.

Chapter 4 proposed a hierarchical behavior planning framework that synergizes low-level safe controllers with a high-level RL coordinator. This approach effectively addressed the complexities of urban traffic conditions, striking a balance between safety and adaptability, which is critical for the real-world application of AVs.

Chapter 5 extended the concept of hierarchical planning to offline RL for temporally extended tasks, showcasing its effectiveness in managing long-duration tasks, a notable challenge in the field.

Chapter 6 explored a novel offline-to-online training scheme, Guided Online Distillation (GOLD), for safety-critical tasks. This approach effectively leveraged large-scale offline data, distilling it into a practical, lightweight policy network suitable for real-time applications.

7.2 Implications and Impact

The work presented in this dissertation has profound implications for the field of autonomous driving and robotics. By addressing critical challenges such as interactive uncertainties, distributional shifts, and the integration of hierarchical planning structures, this research contributes to the development of more reliable, efficient, and safe autonomous systems. The practical applications of these findings extend beyond AVs, offering insights and methodologies that can be applied to a wide range of autonomous systems operating in dynamic and uncertain environments.

7.3 Future Directions and Potential Avenues for Research

While this dissertation has made significant contributions, it also opens several avenues for future research:

- Expanding the scope of data collection to include diverse locations and driving scenarios, enhancing the robustness and adaptability of the developed models.
- Exploring online parameter estimation and adaptive control for real-time behavior adjustment in response to dynamic environmental changes.
- Investigating the application of these methodologies to other scenarios such as intersections, roundabouts, and pedestrian interactions.
- Further integrating persuasion models within the POMDP framework to improve the interactive decision-making process.
- Advancing the hierarchical planning frameworks to accommodate a broader range of scenarios and operational conditions.
- Continuing the development of offline-to-online training schemes like GOLD, focusing on optimizing the balance between performance and computational efficiency.

7.4 Concluding Remarks

In conclusion, this dissertation represents a significant step forward in the field of safe and trustworthy decision-making through reinforcement learning. The methodologies and findings outlined here not only advance our understanding of RL in complex, real-world environments but also lay the groundwork for future innovations in autonomous systems. The intersection of theoretical research and practical application, as demonstrated in this work, will undoubtedly inspire further exploration and development in the quest to create safer, more efficient, and adaptable autonomous technologies.

Bibliography

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. "Constrained policy optimization". In: *International conference on machine learning*. PMLR. 2017, pp. 22–31.

[2] O. Afolabi, K. Driggs–Campbell, R. Dong, M. J. Kochenderfer, and S. S. Sastry. "People as Sensors: Imputing Maps from Human Actions". In: *IROS 2018*. 2018, pp. 2342–2348.

[3] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. "An optimistic perspective on offline reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 104–114.

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

[5] Eitan Altman. "Constrained Markov decision processes with total cost criteria: Lagrangian approach and dual linear program". In: *Mathematical methods of operations research* 48 (1998), pp. 387–417.

[6] Marcin Andrychowicz et al. "Hindsight experience replay". In: *Advances in neural information processing systems* 30 (2017).

[7] Tirthankar Bandyopadhyay, Kok Sung Won, Emilio Frazzoli, David Hsu, Wee Sun Lee, and Daniela Rus. "Intention-Aware Motion Planning". In: *Algorithmic Foundations of Robotics X*. Ed. by Emilio Frazzoli, Tomas Lozano-Perez, Nicholas Roy, and Daniela Rus. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 475–491.

[8] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. "Safe model-based reinforcement learning with stability guarantees". In: *Advances in neural information processing systems* 30 (2017).

[9] Mariusz Bojarski et al. *End to End Learning for Self-Driving Cars*. 2016. arXiv: `1604.07316 [cs.CV]`.

[10] M. Bouton, A. Nakhaei, K. Fujimura, and M. J. Kochenderfer. "Scalable Decision Making with Sensor Occlusions for Autonomous Driving". In: *ICRA 2018*. 2018, pp. 2076–2081.

[11] M. Bouton, A. Nakhaei, K. Fujimura, and M. J. Kochenderfer. “Scalable Decision Making with Sensor Occlusions for Autonomous Driving”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 2076–2081. DOI: `10.1109/ICRA.2018.8460914`.

[12] Kyle Brown, Katherine Driggs-Campbell, and Mykel J. Kochenderfer. “Modeling and Prediction of Human Driver Behavior: A Survey”. In: *arXiv:2006.08832 [cs, eess]* (June 2020).

[13] Tom Brown et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[14] Holger Caesar et al. “nuscenes: A multimodal dataset for autonomous driving”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11621–11631.

[15] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer science & business media, 2013.

[16] Zhangjie Cao, Erdem Bıyık, Woodrow Z. Wang, Allan Raventos, Adrien Gaidon, Guy Rosman, and Dorsa Sadigh. *Reinforcement Learning based Control of Imitative Policies for Near-Accident Driving*. 2020. arXiv: `2007.00178 [cs.LG]`.

[17] Ming-Fang Chang et al. “Argoverse: 3d tracking and forecasting with rich maps”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 8748–8757.

[18] J. Chen, W. Zhan, and M. Tomizuka. “Constrained iterative LQR for on-road autonomous driving motion planning”. In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. 2017, pp. 1–7. DOI: `10.1109/ITSC.2017.8317745`.

[19] Jianyu Chen, Shengbo Eben Li, and Masayoshi Tomizuka. *Interpretable End-to-end Urban Autonomous Driving with Latent Deep Reinforcement Learning*. 2020. arXiv: `2001.08726 [cs.RO]`.

[20] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. *Deep Imitation Learning for Autonomous Driving in Generic Urban Scenarios with Enhanced Safety*. 2019. arXiv: `1903.00640 [cs.RO]`.

[21] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. “Model-free Deep Reinforcement Learning for Urban Autonomous Driving”. In: *CoRR* abs/1904.09503 (2019). arXiv: `1904.09503`. URL: `http://arxiv.org/abs/1904.09503`.

[22] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. “Model-free deep reinforcement learning for urban autonomous driving”. In: *2019 IEEE intelligent transportation systems conference (ITSC)*. IEEE. 2019, pp. 2765–2771.

[23] Kan Chen et al. “WOMD-LiDAR: Raw Sensor Dataset Benchmark for Motion Forecasting”. In: *arXiv preprint arXiv:2304.03834* (Apr. 2023).

[24] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. "Decision transformer: Reinforcement learning via sequence modeling". In: *Advances in neural information processing systems* 34 (2021), pp. 15084–15097.

[25] Yilun Chen, Chiyu Dong, Praveen Palanisamy, Priyantha Mudalige, and et. al. "Attention based hierarchical deep reinforcement learning for lane change behaviors in autonomous driving". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2019.

[26] Yuxin Chen, Chen Tang, Ran Tian, Chenran Li, Jinning Li, Masayoshi Tomizuka, and Wei Zhan. "Quantifying Agent Interaction in Multi-agent Reinforcement Learning for Cost-efficient Generalization". In: *arXiv preprint arXiv:2310.07218* (2023).

[27] Ching-An Cheng, Xinyan Yan, and Byron Boots. "Trajectory-wise control variates for variance reduction in policy gradient methods". In: *Conference on Robot Learning*. PMLR. 2020, pp. 1379–1394.

[28] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc., 2018. URL: `https://proceedings.neurips.cc/paper/2018/file/3de568f8597b94bda53149c7d7f5958c-Paper.pdf`.

[29] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. "End-to-end driving via conditional imitation learning". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 4693–4700.

[30] Alexander Cunningham, Enric Galceran, Ryan Eustice, and Edwin Olson. "MPDM: Multipolicy decision-making in dynamic, uncertain environments for autonomous driving". In: *ICRA 2015*. Vol. 2015. June 2015, pp. 1670–1677. DOI: `10.1109/ICRA.2015.7139412`.

[31] Wojciech M Czarnecki, Razvan Pascanu, Simon Osindero, Siddhant Jayakumar, Grzegorz Swirszcz, and Max Jaderberg. "Distilling policy distillation". In: *The 22nd international conference on artificial intelligence and statistics*. PMLR. 2019, pp. 1331–1340.

[32] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. "A tutorial on the cross-entropy method". In: *Annals of operations research* 134.1 (2005), pp. 19–67.

[33] G. R. de Campos, P. Falcone, and J. Sjöberg. "Autonomous cooperative driving: A velocity-based negotiation approach for intersection crossing". In: *ITSC 2013*. 2013, pp. 1456–1461.

[34] Thomas G Dietterich et al. "The MAXQ Method for Hierarchical Reinforcement Learning." In: *ICML*. Vol. 98. Citeseer. 1998, pp. 118–126.

[35] Wenhao Ding, Mengdi Xu, and Ding Zhao. "Cmts: A conditional multiple trajectory synthesizer for generating safety-critical driving scenarios". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 4314–4321.

[36] Yiming Ding, Carlos Florensa, Mariano Phielipp, and Pieter Abbeel. "Goal conditioned Imitation Learning". In: *CoRR* abs/1906.05838 (2019). arXiv: `1906.05838`. URL: `http://arxiv.org/abs/1906.05838`.

[37] C. Dong, J. M. Dolan, and B. Litkouhi. "Interactive ramp merging planning in autonomous driving: Multi-merging leading PGM (MML-PGM)". In: *ITSC 2017*. 2017, pp. 1–6.

[38] Alexey Dosovitskiy, German Ros, Felipe Codevilla, and et. al. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[39] Jingliang Duan, Shengbo Eben Li, Yang Guan, Qi Sun, and Bo Cheng. "Hierarchical reinforcement learning for self-driving decision-making without reliance on labelled driving data". In: *IET Intelligent Transport Systems* 14.5 (2020), pp. 297–305.

[40] Scott Ettinger et al. "Large Scale Interactive Motion Forecasting for Autonomous Driving: The Waymo Open Motion Dataset". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 9710–9719.

[41] Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. "Search on the replay buffer: Bridging planning and reinforcement learning". In: *Advances in Neural Information Processing Systems* 32 (2019).

[42] Benjamin Eysenbach, Shixiang Gu, Julian Ibarz, and Sergey Levine. "Leave no trace: Learning to reset for safe and autonomous reinforcement learning". In: *arXiv preprint arXiv:1711.06782* (2017).

[43] Angelos Filos, Panagiotis Tigkas, Rowan McAllister, Nicholas Rhinehart, and et. al. "Can autonomous vehicles identify, recover from, and adapt to distribution shifts?" In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3145–3153.

[44] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. *D4RL: Datasets for Deep Data-Driven Reinforcement Learning*. 2020. arXiv: `2004.07219 [cs.LG]`.

[45] Scott Fujimoto, Herke Hoof, and David Meger. "Addressing function approximation error in actor-critic methods". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1587–1596.

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

[47] Sven Gronauer. "Bullet-safety-gym: A framework for constrained reinforcement learning". In: (2022).

[48] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates". In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 3389–3396.

[49] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. "Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning". In: *arXiv preprint arXiv:1910.11956* (2019).

[50] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. arXiv: `1801.01290 [cs.LG]`.

[51] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[52] 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*. PMLR. 2019, pp. 2555–2565.

[53] Hado van Hasselt, Arthur Guez, and David Silver. *Deep Reinforcement Learning with Double Q-learning*. 2015. arXiv: `1509.06461 [cs.LG]`.

[54] Carl-Johan Hoel, Katherine Driggs-Campbell, Krister Wolff, Leo Laine, and Mykel J Kochenderfer. "Combining planning and deep reinforcement learning in tactical decision making for autonomous driving". In: *IEEE Transactions on Intelligent Vehicles* 5.2 (2019), pp. 294–305.

[55] Yeping Hu, Wei Zhan, Liting Sun, and Masayoshi Tomizuka. "Multi-modal Probabilistic Prediction of Interactive Behavior via an Interpretable Model". In: *Intelligent Vehicles Symposium (IV), 2019 IEEE*. June 2019, pp. 557–563.

[56] Jiawei Huang and Nan Jiang. "From importance sampling to doubly robust policy gradient". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 4434–4443.

[57] C. Hubmann, J. Schulz, G. Xu, D. Althoff, and C. Stiller. "A Belief State Planner for Interactive Merge Maneuvers in Congested Traffic". In: *ITSC 2018*. 2018, pp. 1617–1624.

[58] Constantin Hubmann, Jens Schulz, Marvin Becker, Daniel Althoff, and Christoph Stiller. "Automated Driving in Uncertain Environments: Planning with Interaction and Uncertain Maneuver Prediction". In: *IEEE Transactions on Intelligent Vehicles* 3.1 (2018), pp. 5–17.

[59] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. “Imitation learning: A survey of learning methods”. In: *ACM Computing Surveys (CSUR)* 50.2 (2017), pp. 1–35.

[60] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. “When to trust your model: Model-based policy optimization”. In: *arXiv preprint arXiv:1906.08253* (2019).

[61] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. “When to trust your model: Model-based policy optimization”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[62] Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. “Way off-policy batch deep reinforcement learning of implicit human preferences in dialog”. In: *arXiv preprint arXiv:1907.00456* (2019).

[63] Rui Jiang, Qingsong Wu, and Zuojin Zhu. “Full velocity difference model for a car-following theory”. In: *Phys. Rev. E* 64 (1 June 2001), p. 017101. DOI: `10.1103/PhysRevE.64.017101`. URL: `https://link.aps.org/doi/10.1103/PhysRevE.64.017101`.

[64] LI Jinning. “A Novel Integrated SVM for Fault Diagnosis Using KPCA and GA”. In: *Journal of Physics: Conference Series*. Vol. 1207. 1. IOP Publishing. 2019, p. 012002.

[65] Leslie Pack Kaelbling. “Learning to Achieve Goals”. In: *IN PROC. OF IJCAI-93*. Morgan Kaufmann, 1993, pp. 1094–1098.

[66] Gregory Kahn, Pieter Abbeel, and Sergey Levine. “Badgr: An autonomous self-supervised learning-based navigation system”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 1312–1319.

[67] Dmitry Kalashnikov et al. “Scalable deep reinforcement learning for vision-based robotic manipulation”. In: *Conference on Robot Learning*. PMLR. 2018, pp. 651–673.

[68] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. “MOReL: Model-Based Offline Reinforcement Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 21810–21823.

[69] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli. “Kinematic and dynamic vehicle models for autonomous driving control design”. In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. 2015, pp. 1094–1099. DOI: `10.1109/IVS.2015.7225830`.

[70] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. “Offline reinforcement learning with implicit q-learning”. In: *arXiv preprint arXiv:2110.06169* (2021).

[71] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. “Stabilizing off-policy q-learning via bootstrapping error reduction”. In: *arXiv preprint arXiv:1906.00949* (2019).

[72] Aviral Kumar, Joey Hong, Anikait Singh, and Sergey Levine. "When should we prefer offline reinforcement learning over behavioral cloning?" In: *arXiv preprint arXiv:2204.05618* (2022).

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

[74] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. "Conservative q-learning for offline reinforcement learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1179–1191.

[75] Hanna Kurniawati and Vinay Yadav. "An Online POMDP Solver for Uncertainty Planning in Dynamic Environment". In: Apr. 2016, pp. 611–629. ISBN: 978-3-319-28870-3. DOI: 10.1007/978-3-319-28872-7_35.

[76] Pawel Ladosz, Lilian Weng, Minwoo Kim, and Hyondong Oh. "Exploration in deep reinforcement learning: A survey". In: *Information Fusion* 85 (2022), pp. 1–22.

[77] Edouard Leurent and Jean Mercat. *Social Attention for Autonomous Decision-Making in Dense Traffic*. 2019. arXiv: 1911.12250 [cs.LG].

[78] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. "Offline reinforcement learning: Tutorial, review, and perspectives on open problems". In: *arXiv preprint arXiv:2005.01643* (2020).

[79] Jiachen Li, Hengbo Ma, and Masayoshi Tomizuka. "Conditional Generative Neural System for Probabilistic Trajectory Prediction". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 6150–6156.

[80] Jiachen Li, Hengbo Ma, Zhihao Zhang, Jinning Li, and Masayoshi Tomizuka. "Spatio-temporal graph dual-attention network for multi-agent prediction and tracking". In: *IEEE Transactions on Intelligent Transportation Systems* 23.8 (2021), pp. 10556–10569.

[81] Jiachen Li, Fan Yang, Masayoshi Tomizuka, and Chiho Choi. "EvolveGraph: Multi-Agent Trajectory Prediction with Dynamic Relational Reasoning". In: *2020 Advances in Neural Information Processing Systems (NeurIPS)*. 2020.

[82] Jinning Li, Xinyi Liu, Banghua Zhu, Jiantao Jiao, Masayoshi Tomizuka, Chen Tang, and Wei Zhan. "Guided online distillation: Promoting safe reinforcement learning by offline demonstration". In: *arXiv preprint arXiv:2309.09408* (2023).

[83] Jinning Li, Liting Sun, Jianyu Chen, Masayoshi Tomizuka, and Wei Zhan. "A safe hierarchical planning framework for complex driving scenarios based on reinforcement learning". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 2660–2666.

[84] Jinning Li, Liting Sun, Wei Zhan, and Masayoshi Tomizuka. “Interaction-aware behavior planning for autonomous vehicles validated with real traffic data”. In: *Dynamic Systems and Control Conference*. Vol. 84287. American Society of Mechanical Engineers. 2020, V002T31A005.

[85] Jinning Li, Chen Tang, Masayoshi Tomizuka, and Wei Zhan. “Dealing with the Unknown: Pessimistic Offline Reinforcement Learning”. In: *Conference on Robot Learning*. PMLR. 2022, pp. 1455–1464.

[86] Jinning Li, Chen Tang, Masayoshi Tomizuka, and Wei Zhan. “Hierarchical Planning Through Goal-Conditioned Offline Reinforcement Learning”. In: *arXiv preprint arXiv:2205.11790* (2022).

[87] Quanyi Li, Zhenghao Peng, Lan Feng, Qihang Zhang, Zhenghai Xue, and Bolei Zhou. “Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022).

[88] Zuxin Liu, Zhepeng Cen, Vladislav Isenbaev, Wei Liu, Steven Wu, Bo Li, and Ding Zhao. “Constrained variational policy optimization for safe reinforcement learning”. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 13644–13668.

[89] Zuxin Liu, Zijian Guo, Zhepeng Cen, Huan Zhang, Jie Tan, Bo Li, and Ding Zhao. “On the robustness of safe reinforcement learning under observational perturbations”. In: *arXiv preprint arXiv:2205.14691* (2022).

[90] Chaoru Lu and Arvid Aakre. “A new adaptive cruise control strategy and its stabilization effect on traffic flow”. In: *European Transport Research Review* (2018). DOI: `https://doi.org/10.1186/s12544-018-0321-9`.

[91] H. Ma, J. Li, W. Zhan, and M. Tomizuka. “Wasserstein Generative Learning with Kinematic Constraints for Probabilistic Interactive Driving Behavior Prediction”. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. June 2019, pp. 2477–2483.

[92] Ajay Mandlekar, Fabio Ramos, Byron Boots, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Dieter Fox. “Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 4414–4420.

[93] Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín, Silvio Savarese, and Li Fei-Fei. “Learning to generalize across long-horizon tasks from human demonstrations”. In: *arXiv preprint arXiv:2003.06085* (2020).

[94] Ajay Mandlekar et al. “What matters in learning from offline human demonstrations for robot manipulation”. In: *arXiv preprint arXiv:2108.03298* (2021).

[95] Matthew McNaughton, Chris Urmson, John M Dolan, and Jin-Woo Lee. “Motion planning for autonomous driving with a conformal spatiotemporal lattice”. In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 4889–4895.

[96] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. "Data-efficient hierarchical reinforcement learning". In: *Advances in neural information processing systems* 31 (2018).

[97] Mitsuhiko Nakamoto, Yuexiang Zhai, Anikait Singh, Max Sobol Mark, Yi Ma, Chelsea Finn, Aviral Kumar, and Sergey Levine. "Cal-ql: Calibrated offline rl pre-training for efficient online fine-tuning". In: *arXiv preprint arXiv:2303.05479* (2023).

[98] Soroush Nasiriany, Vitchyr H Pong, Steven Lin, and Sergey Levine. "Planning with goal-conditioned policies". In: *arXiv preprint arXiv:1911.08453* (2019).

[99] Tomoki Nishi, Prashant Doshi, and Danil Prokhorov. "Merging in Congested Freeway Traffic Using Multipolicy Decision Making and Passive Actor-Critic Learning". In: *IEEE Transactions on Intelligent Vehicles* 4.2 (June 2019), pp. 287–297. ISSN: 2379-8858. DOI: 10.1109/tiv.2019.2904417. URL: http://dx.doi.org/10.1109/TIV.2019.2904417.

[100] Masoud S Nosrati, Elmira Amirloo Abolfathi, Mohammed Elmahgiubi, Peyman Yadmellat, Jun Luo, Yunfei Zhang, Hengshuai Yao, Hongbo Zhang, and Anas Jamil. "Towards practical hierarchical reinforcement learning for multi-lane autonomous driving". In: (2018).

[101] Brendan O'Donoghue, Ian Osband, Remi Munos, and Volodymyr Mnih. "The uncertainty bellman equation and exploration". In: *International Conference on Machine Learning*. 2018, pp. 3836–3845.

[102] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. "Deep exploration via bootstrapped DQN". In: *arXiv preprint arXiv:1602.04621* (2016).

[103] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. "A survey of motion planning and control techniques for self-driving urban vehicles". In: *IEEE Transactions on intelligent vehicles* 1.1 (2016), pp. 33–55.

[104] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. *Virtual to Real Reinforcement Learning for Autonomous Driving*. 2017. arXiv: 1704.03952 [cs.AI].

[105] Sergey Pankov. "Reward-estimation variance elimination in sequential decision processes". In: *arXiv preprint arXiv:1811.06225* (2018).

[106] C. Peng and M. Tomizuka. "Bayesian Persuasive Driving". In: *2019 American Control Conference (ACC)*. 2019, pp. 723–729. DOI: 10.23919/ACC.2019.8814319.

[107] Cheng Peng and Masayoshi Tomizuka. "Bayesian Persuasive Driving". In: *ACC 2019*. IEEE, July 2019.

[108] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. "Advantage-weighted regression: Simple and scalable off-policy reinforcement learning". In: *arXiv preprint arXiv:1910.00177* (2019).

[109] Dean A Pomerleau. "Alvinn: An autonomous land vehicle in a neural network". In: *Advances in neural information processing systems*. 1989, pp. 305–313.

[110] Doina Precup. “Eligibility traces for off-policy policy evaluation”. In: *Computer Science Department Faculty Publication Series* (2000), p. 80.

[111] Ali Punjani and Pieter Abbeel. “Deep learning helicopter dynamics models”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 3223–3230.

[112] Zhiqian Qiao, Zachariah Tyree, Priyantha Mudalige, Jeff Schneider, and John M. Dolan. *Hierarchical Reinforcement Learning Method for Autonomous Vehicle Behavior Planning*. 2019. arXiv: `1911.03799 [cs.RO]`.

[113] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. “Improving language understanding by generative pre-training”. In: (2018).

[114] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. “Learning complex dexterous manipulation with deep reinforcement learning and demonstrations”. In: *arXiv preprint arXiv:1709.10087* (2017).

[115] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. “Zero-shot text-to-image generation”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8821–8831.

[116] Tabish Rashid, Bei Peng, Wendelin Boehmer, and Shimon Whiteson. “Optimistic exploration even with a pessimistic initialisation”. In: *arXiv preprint arXiv:2002.12174* (2020).

[117] Paria Rashidinejad, Banghua Zhu, Cong Ma, Jiantao Jiao, and Stuart Russell. “Bridging Offline Reinforcement Learning and Imitation Learning: A Tale of Pessimism”. In: *CoRR* abs/2103.12021 (2021). arXiv: `2103.12021`. URL: `https://arxiv.org/abs/2103.12021`.

[118] Alex Ray, Joshua Achiam, and Dario Amodei. “Benchmarking Safe Exploration in Deep Reinforcement Learning”. In: (2019).

[119] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. “Policy distillation”. In: *arXiv preprint arXiv:1511.06295* (2015).

[120] Dorsa Sadigh, Shankar Sastry, Sanjit Seshia, and Anca Dragan. “Planning for Autonomous Cars that Leverage Effects on Human Actions”. In: *RSS 2016*. June 2016.

[121] Dorsa Sadigh, Shankar Sastry, Sanjit A Seshia, and Anca D Dragan. “Planning for autonomous cars that leverage effects on human actions.” In: *Robotics: Science and Systems*. Vol. 2. Ann Arbor, MI, USA. 2016.

[122] AhmadEL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. "Deep Reinforcement Learning framework for Autonomous Driving". In: *Electronic Imaging* 2017.19 (Jan. 2017), pp. 70–76. ISSN: 2470-1173. DOI: `10.2352/issn.2470-1173.2017.19.avm-023`. URL: `http://dx.doi.org/10.2352/ISSN.2470-1173.2017.19.AVM-023`.

[123] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. "Universal Value Function Approximators". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1312–1320. URL: `https://proceedings.mlr.press/v37/schaul15.html`.

[124] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms". In: *arXiv preprint* (2017).

[125] M. Sefati, J. Chandiramani, K. Kreiskoether, A. Kampker, and S. Baldi. "Towards tactical behaviour planning under uncertainties for automated vehicles in urban scenarios". In: *ITSC 2017*. Oct. 2017, pp. 1–7.

[126] Shahaf S Shperberg, Bo Liu, and Peter Stone. "Relaxed Exploration Constrained Reinforcement Learning". In: *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*. 2023, pp. 2821–2823.

[127] Noah Y Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin Riedmiller. "Keep doing what worked: Behavioral modelling priors for offline reinforcement learning". In: *arXiv preprint arXiv:2002.08396* (2020).

[128] Avi Singh, Albert Yu, Jonathan Yang, Jesse Zhang, Aviral Kumar, and Sergey Levine. "Cog: Connecting new skills to past experience with offline reinforcement learning". In: *arXiv preprint arXiv:2010.14500* (2020).

[129] Trey Smith and Reid Simmons. "Point-based POMDP algorithms: Improved analysis and implementation". In: *arXiv preprint arXiv:1207.1412* (2012).

[130] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. "Learning structured output representation using deep conditional generative models". In: *Advances in neural information processing systems* 28 (2015).

[131] Liting Sun, Cheng Peng, Wei Zhan, and Masayoshi Tomizuka. "A fast integrated planning and control framework for autonomous driving via imitation learning". In: *Dynamic Systems and Control Conference*. Vol. 51913. American Society of Mechanical Engineers. 2018, V003T37A012.

[132] Liting Sun, Wei Zhan, Ching-Yao Chan, and Masayoshi Tomizuka. "Behavior Planning of Autonomous Cars with Social Perception". In: *IEEE Intelligent Vehicles Symposium*. 2019.

[133] Yu Sun, Wyatt L Ubellacker, Wen-Loong Ma, Xiang Zhang, Changhao Wang, Noel V Csomay-Shanklin, Masayoshi Tomizuka, Koushil Sreenath, and Aaron D Ames. "Online learning of unknown dynamics for model-based controllers in legged locomotion". In: *IEEE Robotics and Automation Letters* 6.4 (2021), pp. 8442–8449.

[134] Richard S Sutton. "Dyna, an integrated architecture for learning, planning, and reacting". In: *ACM Sigart Bulletin* 2.4 (1991), pp. 160–163.

[135] Brijen Thananjeyan, Ashwin Balakrishna, Ugo Rosolia, Felix Li, Rowan McAllister, Joseph E Gonzalez, Sergey Levine, Francesco Borrelli, and Ken Goldberg. "Safety Augmented Value Estimation from Demonstrations (SAVED): Safe Deep Model-Based RL for Sparse Cost Robotic Tasks". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3612–3619.

[136] René Traoré, Hugo Caselles-Dupré, Timothée Lesort, Te Sun, Guanghang Cai, Natalia Díaz-Rodríguez, and David Filliat. "Discorl: Continual reinforcement learning via policy distillation". In: *arXiv preprint arXiv:1907.05855* (2019).

[137] M. Treiber, A. Hennecke, and D. Helbing. "Congested Traffic States in Empirical Observations and Microscopic Simulations". In: *Physical Review E (Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics)* 62 (2000), pp. 1805–1824. DOI: `10.1103/PhysRevE.62.1805`.

[138] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. "Congested traffic states in empirical observations and microscopic simulations". In: *Physical Review E* 62.2 (Aug. 2000), pp. 1805–1824. ISSN: 1095-3787. DOI: `10.1103/physreve.62.1805`. URL: `http://dx.doi.org/10.1103/PhysRevE.62.1805`.

[139] Ikechukwu Uchendu et al. "Jump-start reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2023, pp. 34556–34583.

[140] S. Ulbrich and M. Maurer. "Probabilistic online POMDP decision making for lane changes in fully automated driving". In: *ITSC 2013*. 2013, pp. 2063–2067.

[141] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 2016.

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

[143] Changhao Wang, Yuyou Zhang, Xiang Zhang, Zheng Wu, Xinghao Zhu, Shiyu Jin, Te Tang, and Masayoshi Tomizuka. "Offline-online learning of deformation model for cable manipulation with graph neural networks". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 5544–5551.

[144] Jingke Wang, Yue Wang, Dongkun Zhang, Yezhou Yang, and Rong Xiong. *Learning hierarchical behavior and motion planning for autonomous driving*. 2020. arXiv: `2005.03863 [cs.RO]`.

[145] P. Wang, H. Li, and C. Chan. "Continuous Control for Automated Lane Change Behavior Based on Deep Deterministic Policy Gradient Algorithm". In: *IV 2019*. 2019, pp. 1454–1460.

[146] Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin Riedmiller. "Embed to control: A locally linear latent dynamics model for control from raw images". In: *arXiv preprint arXiv:1506.07365* (2015).

[147] Yifan Wu, George Tucker, and Ofir Nachum. "Behavior regularized offline reinforcement learning". In: *arXiv preprint arXiv:1911.11361* (2019).

[148] Zheng Wu, Wenzhao Lian, Changhao Wang, Mengxi Li, Stefan Schaal, and Masayoshi Tomizuka. "Prim-LAfD: A Framework to Learn and Adapt Primitive-Based Skills from Demonstrations for Insertion Tasks". In: *arXiv preprint arXiv:2212.00955* (2022).

[149] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. *End-to-end Learning of Driving Models from Large-scale Video Datasets*. 2017. arXiv: `1612.01079 [cs.CV]`.

[150] Ruochen Yang and Paul Bogdan. "Controlling the multifractal generating measures of complex networks". In: *Scientific reports* 10.1 (2020), p. 5541.

[151] Ruochen Yang, Gaurav Gupta, and Paul Bogdan. "Data-driven perception of neuron point process with unknown unknowns". In: *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*. 2019, pp. 259–269.

[152] Ruochen Yang, Frederic Sala, and Paul Bogdan. "Hidden network generating rules from partially observed complex networks". In: *Communications Physics* 4.1 (2021), p. 199.

[153] W. Zhan, A. L. de Fortelle, Y. Chen, C. Chan, and M. Tomizuka. "Probabilistic Prediction from Planning Perspective: Problem Formulation, Representation Simplification and Evaluation Metric". In: *Intelligent Vehicles Symposium (IV), 2018 IEEE*. June 2018, pp. 1150–1156.

[154] Wei Zhan, Liting Sun, Yeping Hu, Jiachen Li, and Masayoshi Tomizuka. "Towards a fatality-aware benchmark of probabilistic reaction prediction in highly interactive driving scenarios". In: *ITSC 2018*. IEEE. 2018, pp. 3274–3280.

[155] Wei Zhan, Liting Sun, Di Wang, Yinghan Jin, and Masayoshi Tomizuka. "Constructing a highly interactive vehicle motion dataset". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 6415–6420.

[156] Wei Zhan et al. "INTERACTION Dataset: An INTERnational, Adversarial and Cooperative moTION Dataset in Interactive Driving Scenarios with Semantic Maps". In: *arXiv:1910.03088 [cs, eess]* (2019).

[157] Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew Johnson, and Sergey Levine. "Solar: Deep structured representations for model-based reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 7444–7453.

[158] Xiang Zhang, Changhao Wang, Lingfeng Sun, Zheng Wu, Xinghao Zhu, and Masayoshi Tomizuka. “Efficient Sim-to-real Transfer of Contact-Rich Manipulation Skills with Online Admittance Residual Learning”. In: *7th Annual Conference on Robot Learning*. 2023.

[159] Wenxuan Zhou, Sujay Bajracharya, and David Held. “PLAS: Latent Action Space for Offline Reinforcement Learning”. In: *arXiv preprint arXiv:2011.07213* (2020).