Driving by Dreaming: Offline Model-Based Reinforcement Learning for Autonomous Vehicles

Swapnil Pande

CMU-RI-TR-22-49

August 5, 2022

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

Thesis Committee:
Jeff Schneider, *chair*
Deva Ramanan
David Held
Ian Char

Submitted in partial fulfillment of the requirements for the degree of Masters in Robotics.

To my family, friends, and mentors who have served as role models and supported me through this journey.

Abstract

While there has been significant progress in deploying autonomous vehicles (AVs) in urban driving settings, there remains a long-tail of challenging motion planning scenarios that must be addressed before truly driverless operation is possible. The current paradigm for motion planner design is engineering intensive, making it challenging to scale to address the long-tail. In this work, we explore the use of offline model-based reinforcement learning as an alternative approach for designing AV motion planners. We propose an offline RL algorithm which make use of the structure in the AV domain to create dynamics models and policies that generalize successfully. Additionally, we propose an extension to this algorithm to a multi-agent training environment. We demonstrate that these algorithms can match state-of-the-art performance on simpler driving benchmarks and explore future works for scaling to the more challenging benchmarks.

Acknowledgments

There are countless people who have inspired and supported me throughout this journey. I would like to take this opportunity to thank a select few.

First, I'd like to thank my advisor, Professor Jeff Schneider, for his incredible support over the past two years. His guidance and insight on this project has been invaluable in helping me find interesting research avenues and making this project successful. I have grown significantly as a researcher, which he made possible through his thought-provoking discussions, flexibility in allowing me to explore my interests, and help in overcoming roadblocks.

Further, I would like thank my other committee members, Professor Deva Ramanan, Professor David Held, and Ian Char. Their feedback throughout the process of assembling this thesis has helped strengthen this work and has provided me different perspectives on how this work connects to the broader work in this domain, which has deepened my understanding of the field.

This research was not done in isolation, but rather was supported by the many members of the Auton lab. I'd like to particularly express my gratitude to Brian Yang, Yeeho Song, Adam Villaflor, Zhe Huang, Viraj Mehta, Ian Char, and Conor Igoe, who have all engaged me in stimulating conversation and helped create an enjoyable work environment.

My family and friends have provided me unconditional support through the thick and thin of the past two years. I would like to particularly thank Benjamin Freed, who has been an incredible mentor in helping me discover my research interests and grappling with the challenges of academia. Along with Ben, Mateo Guaman Castro, Alex Stephens, Raunaq Bhirangi, Govind Pande, Nutan Pande, and Apurva Pande have been immeasurably supportive, helping pick me up through my hardships and celebrate my victories with me

Finally, I would like to thank Dr. Predrag Punosevac, who maintained the Auton compute cluster throughout this work. All of the behind-the-scenes work to keep the cluster operational made this research possible, and I am very grateful for it.

To all of these people and the many more who have been by my side, thank you so much for your unwavering support. You have made this

journey enjoyable and have challenged me to grow to my best self.

Funding

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

Contents

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

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Current Paradigm in Autonomous Vehicle Industry

Over the past 10 years, the goal of autonomous vehicles (AVs) for urban driving has evolved from a purely academic vision to a large focus of the auto industry. With the large advances in computer vision driven by the adoption of convolutional neural networks, there was strong sentiment within the industry that fully driverless operation was technologically within reach and simply required scale to achieve. However, after many years of intensive engineering effort, the industry has realized that there still exist many unsolved problems that need to be addressed before we can truly scale to full driverless operation.

Particularly, a problem whose magnitude was under-appreciated in the industry is that of motion planning. Even with access to ground-truth state information from a "perfect" vision stack, there is a very long-tail of challenging scenarios in day-to-day urban driving that require a multitude of strategies and challenging contextual reasoning to solve. Often times, a patch to a motion planner to address a particular challenging scenario often directly reduces performance on another long-tail scenario. Thus, it remains unclear what the best strategy is for addressing this long-tail of scenarios.

Currently, most autonomous vehicle companies in industry follow a software

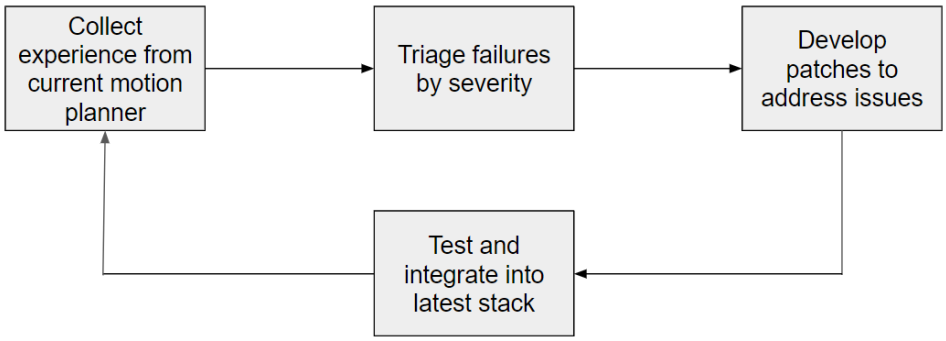

Figure 1.1: A typical Motion Planner iteration cycle for autonomous vehicle companies. Failures from the current version of the planner go through a triage process and get addressed as patches. The triage and patch development process and very engineering intensive and are challenging to scale

development workflow similar to the one depicted in Figure 1.1 . A triage team will review driving logs and identify which portion of the stack likely caused a given failure. These failures are passed onto the relevant team, who develop new features or patches to address the issues. The patches are then run through an extensive series of unit tests and regression tests to evaluate performance on key driving metrics, before they get integrated with the production AV stack. Independently, teams also work torwards large feature upgrades that hope to handle these issues in a more holistic/principled way. However, this system of small incremental changes leads to an overly complex stack with suboptimal choices that become challenging to reverse. Again, in the case of motion planning, addressing issues in this manner is very engineering intensive due to the sheer number of challenging scenarios encountered during driving.

1.1.1 Machine Learning for Motion Planning

Drawing from the huge successes of deep learning for computer vision on autonomous vehicles, there has been a significant growth in interest in applying deep learning techniques to the motion planning domain. The direct analog to the end-to-end supervised vision approaches is imitation learning, in which we train a policy to copy the actions in expert demonstrations. In the case of self-driving, performing imitation learning would require a very large dataset of human driving demonstrations, which is expensive to collect. Further, the performance of a policy trained via imitation

learning is upper bounded by the performance of the data-collection agents, which ultimately does not address the goal of creating AVs that are safer than human drivers. Another option is to use deep reinforcement learning (RL) algorithms, which optimize a policy to generate actions that maximize a pre-defined reward function. In the online formulation of reinforcement learning, which has seen strong success in literature, the policy is trained by deploying it environment to collect experience, using that experience to update the policy, and re-deploying the policy in a loop. In many robotics problems, including driving, collecting data online is usually intractable due to safety concerns of deploying an untrained policy and because data collection becomes inefficient with frequent disengagements due to policy failures. Therefore, another commonly studied approach in robotics is to use high-fidelity simulators for training, allowing for faster and safer data collection. Depending on the complexity of the environment, designing and calibrating a high-fidelity simulator is an engineering intensive task that quickly becomes intractable. In driving specifically, while we can build strong models of vehicle dynamics, it is difficult to program a wide distribution of behaviors for the other vehicles, thereby making it expensive to generate challenging long-tail scenarios.

In this work, we focus applying Offline Model-Based Reinforcement Learning (Offline MBRL) for learning a motion planning policy as a potential alternative to the current motion planning paradigm. In the offline setting, we aim to learn a policy from a pre-collected dataset of driving experience, instead of collecting data with the policy online. The policy is only deployed in the environment during test time. This is particularly attractive in the AV industry as all companies already collect large datasets of driving experience that can be used for training. In model-based RL, we additionally learn a dynamics model to predict how the environment evolves given the policy actions, which can be used for policy optimization. Learning a dynamics model is attractive as the policy can generate trajectories not present in the dataset and receive a reward signal about them. Again, this is valuable for AV motion planning as it allows the policy to learn from challenging scenarios that are not explicitly demonstrated in the dataset. However, offline model-based algorithms are accompanied with many issues as well. For example, if a policy strays too far from the trajectories presented in the dataset, improper model generalization can cause the policy to learn behaviors that could be catastrophic when deployed at test

time.

With offline RL, we have the potential to re-imagine the motion planning iteration cycle in an AV stack. However, there are many issues that need to be addressed before it can be used as truly valuable tool in the AV industry. In this work, we present a series of algorithms that apply offline MBRL for learning motion planning policies for autonomous vehicles. We explore the limitations of current offline MBRL algorithms and improvements that significantly improve their performance in AV motion planning. We propose an offline MBRL algorithm that makes use of these improvements and an extension of this algorithm to a multi-agent training setup. We study the advantages and limitations of these algorithms, and propose a vision for how this could be integrated into a real AV stack.

Chapter 2

Background

2.1 Reinforcement Learning

2.1.1 Preliminaries

We consider the task of motion planning for AVs as a Partially-Observable Markov Decision process (POMDP), defined as the tuple $(S, O, A, r, P, \rho_0, \gamma)$, where S is the state space, O is the observation space, $r : S \times A$ is the reward function, $P : S \times A \times S$ is the state transition probability, ρ_0 is the initial state distribution, and γ is the discount factor. Our objective is to find a policy $\pi^*(a_t|o_t)$ that maximizes the expected discounted returns.

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right] \tag{2.1}$$

2.1.2 Imitation Learning

Imitation learning (IL) attempts to find an optimal policy for a given Markov Decision Process (MDP) simply by replicating the behavior of an expert. One of the simplest forms of imitation learning is behavior cloning, in which the policy is trained by empirical risk minimization on a dataset of actions generated by the demonstrator. A key limitation of IL algorithms is that their performance is bounded by the performance

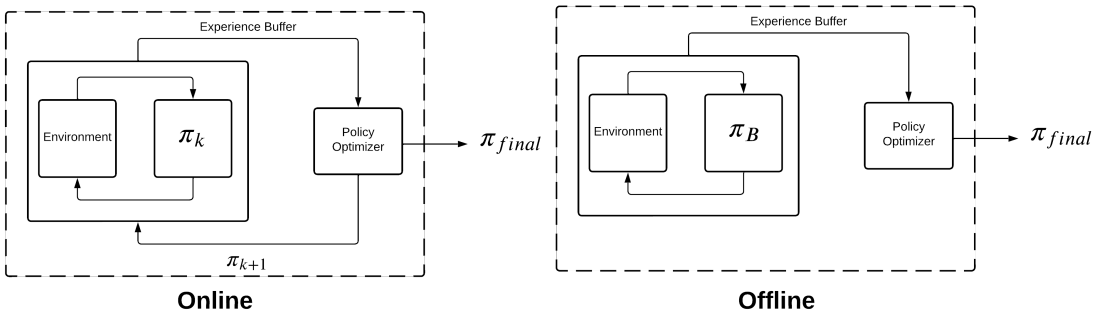

Figure 2.1: A comparison of the training procedure between online and offline reinforcement learning. In the online setting, the most recent policy is used to collect experience from the environment. The experience is used to update the policy, which is then re-deployed to collect more data. In the offline setting, a behavioral policy collects data from the environment once and that data is directly used to optimize a policy without collecting new data.

of the expert demonstrations. This is particularly limiting in driving because we do not have access to an oracle policy. While humans can provide strong demonstrations, the goal in the autonomous vehicle industry is to ultimately outperform human drivers. Even if we assume access to demonstrations from an oracle policy, [15] shows that the error bound for behavior cloning between the oracle and the learned policy grows quadratically with the MDP horizon due to the issue of distribution shift from the static training dataset. These upper bounds on performance ultimately limit the effectiveness of imitation learning for the AV problem.

2.1.3 Online and Offline Reinforcement Learning

A comparison of the online and offline reinforcement learning settings can be found in Figure 2.1. In the online reinforcement learning setting, the policy being optimized can interact with the environment for which is it being trained. Most online RL algorithms employ a training loop that involves collecting experience by deploying the policy in the environment, performing weight updates on the policy given the experience, and repeating the cycle to collect new experience. In the Offline RL setting, on the other hand, the policy cannot be deployed in the environment to collect data during training time. Instead, Offline RL assumes access to a dataset $\mathcal{D} = \{s_i, o_i, a_i, r_i, s_{i+1}\}_{i=1}^{N}$, generated by a set of data generation policies π_D that we

do not have access to.

2.1.4 Model-Free and Model-Based Reinforcement Learning

Another distinguishing feature in RL algorithms is whether they are model-free or model-based. In model-based RL, we assume access to, or learn, a world transition model that can be used directly for online planning or for policy optimization. In model-free RL, the algorithm does not have access to a transition model and instead directly optimizes policies to maximize the reward function. Model-based RL algorithms tend to be quite sample efficient, while model-free algorithms tend to demonstrate strong asymptotic performance.

2.2 Related Works

2.2.1 Learning for Self-Driving Cars

The majority of the literature in motion planning for autonomous vehicles focuses on imitating expert demonstrations [3, 4, 5]. In [3] and [5], the demonstrations are generated by hand-crafted autopilot policies. These works focus heavily on the representation learning problem of mapping raw sensor data to control actions. While these methods have shown success, these methods are difficult to scale as they require expert demonstrations in a large set of challenging scenarios. We instead choose to focus on learning policies without the need for expert demonstrations instead of addressing the representation learning problem.

Recently with the successes of deep reinforcement learning (DRL) for a wide variety of planning and control tasks, DRL, reinforcement learning methods have begun to surpass the performance of imitation agents in standard self-driving benchmarks. Particularly, [1, 9] demonstrate strong performance on the standard driving benchmarks, using policies trained online with Proximal Policy Optimization (PPO) [19] and Soft Actor Critic (SAC). These works consider a hand-crafted low-dimensional state space that contains the necessary features for learning effective driving behaviors, and also demonstrate success learning directly from semantically segmented images. Formulating the problem as a sequential decision making process is better

suited to self-driving compared to treating each time-step as independent as imitation learning does. We adopt a slightly modified version of the low-dimensional state-space presented in this work and apply it to learning in the offline setting.

We make use of the CARLA simulator [7] to construct our RL environment. The simulator is a photo-realistic simulator with many tools designed for autonomous vehicle research, including pre-defined sensors, heuristic autopilot policies, and various urban driving maps. Additionally, we make use of the pre-defined benchmarks defined for the CARLA simulator to evaluate our policies.

2.2.2 Offline Reinforcement Learning

There has been growing interest in developing and improving reinforcement learning algorithms for the offline setting. One group of methods for Offline RL focus on improving the stability of off-policy Q-learning by reducing the overestimation of the Q-function in regions outside the support of the dataset [13, 14]. While these methods have proven to be successful, we focus on model-based offline RL because of its potential to "dream" of rare scenarios not present in the original dataset, such as near collisions with other vehicles and unpredictable pedestrians.

Another class of methods focus on performing offline RL in a model-based setting with a learning procedure similar to the one presented in [11]. At a high level, these methods first optimize a model $f(s_t, a_t)$ to predict the transition dynamics of the environment. They then train a policy by performing autoregressive rollouts of the dynamics model with actions sampled from the policy being optimized. However, similar to the Q-learning algorithms, the model often poorly extrapolates in regions of the state space outside of the data distribution. Therefore, recent works such as [12, 22] present uncertainty-aware dynamics models and introduce a penalty in the reward function or termination function conditioned on the state estimation uncertainty during the policy optimization. We focus on applying an algorithm similar to those presented in [12, 22], in which the policy is penalized proportionally to the magnitude of the model's uncertainty in the given state. However, we improve upon their dynamics model structure, by building a model that encodes inductive biases about the structure of the AV motion planning problem.

2.2.3 Multi-Agent Reinforcement Learning

The area of multi-agent RL has seen many big successes, such as AlphaStar [21] and OpenAI Five [18]. Both of these works train agents in a self-play fashion, in which they play against themselves as adversaries. AlphaStar achieved grandmaster level performance in Starcraft and OpenAI Five achieved world champion level performance in Dota 2, both learning strategies for these games that were novel to their human opponents. These works demonstrate the potential of multi-agent RL to scale to large problems with massive datasets. However, these environments differ drastically from the driving problem as they are fully competitive zero-sum games. Other works focus on fully cooperative environments, such as in the game Hanabi as proposed in [2]. Motion Planning for AV is neither fully competitive, nor fully cooperative, making it a challenging problem for which to build a multi-agent training algorithm.

Chapter 3

Problem Formulation

3.1 CARLA Simulator

3.1.1 Introduction

We make extensive use of the CARLA Simulator [7] to train and evaluate our algorithms. CARLA is a photo-realistic, open-source urban, driving simulator built with Unreal Engine for the purpose of AV research. The simulator defines many features such as standard AV sensors, multiples "towns" with diverse urban features and road layouts, and heuristic autopilot policies for non-playable vehicles and pedestrians. Due to the ability to access ground-truth state of the simulator, CARLA facilitates research on individual components, such as motion planning, without requiring the development of the entire stack to process raw sensor data. We build all of our work on CARLA 9.10 and run the simulator at 10 Hz.

3.1.2 Benchmarks

The CARLA community has also defined a set of benchmarks built on top of CARLA to evaluate the performance of motion planning algorithms.

NoCrash

The NoCrash Benchmark [6] defines a set of 25 training routes in Town01 and 25 testing routes in Town02. The three levels of the benchmark, Empty, Regular, and Dense, vary in the number of other vehicles spawned in the town. An episode is considered a success if the agent successfully completes the route without colliding with any vehicle or static obstacle. The benchmark does not explicitly penalize the agent for not obeying traffic lights or exiting the lane, though not obeying these makes a high score on the Regular and Dense levels challenging to achieve.

Leaderboard

The Leaderboard benchmark[1] represents a significant increase in difficulty over the NoCrash benchmark. Similar to NoCrash, Leaderboard also defines a set of routes to follow, but includes routes on Towns 02-06 as well. The routes are significantly longer than those in NoCrash and spawn a higher density of vehicles and pedestrians. More importantly, Leaderboard introduces ten common pre-crash scenarios defined by NHSTA[2] that spawned as the vehicle reaches checkpoints along the route. Some illustrative classes of scenarios are listed below and the complete list of scenarios can be found on the Leaderboard challenge website[3].

- Control loss due to slippery conditions
- Lane changes around vehicles or static objects
- Merging onto highways
- Negotiating intersections with actors not obeying traffic rules

In addition to defining more challenging scenarios, the benchmark also restricts the type of data the policy has access to for planning. They define a limited set of sensors and pseudo-sensors that can be mounted on the AV. Futhermore, they only provide sparse guidance about the route to follow in the form of high-level navigation commands such as `Lane Change Left`, `Lane Follow`, or `Merge Left`. As we detail fully in Section 3.2.1, we relax these requirements to allow us to focus more directly

[1] https://leaderboard.carla.org/
[2] https://nhtsa.gov
[3] https://leaderboard.carla.org/scenarios/

on the motion planning problem, without having to jointly build approaches for perception and global planning.

To better characterize the performance of the ego-vehicle, the Leaderboard benchmark scores performance with a driving score, which is computed as a linear combination of key performance metrics such as route completion, number of collisions, and number of other driving infractions. The full driving score formulation can be found on the Leaderboard website [4]. The benchmark defines 4 sets of routes: `devtest`, `training`, `evaluation`, and `testing`, whose details can be found at [9]. The `devtest`, `training`, and `evaluation` routes are made publicly available, while the `testing` routes are held out for evaluation in the Leaderboard challenge. As we relax the requirements of the benchmark, we do not evaluate on the testing routes.

3.2 MDP Formulation

Below, we present our formulation of the MDP for AV motion planning. Our choices for observation space, action space, and reward function are heavily inspired by [1].

3.2.1 Observation Space

We consider two observation spaces for this problem, both of which consist of four main classes of features: ego-vehicle state, other dynamic actor state, global planner state, and traffic control state. As mentioned previously, we choose to focus independently on the motion planning problem and therefore, assume ground-truth access to simulator state. However, we do only assume access to state information that would realistically be computed in a typical AV stack in the perception, motion forecasting, state estimation, and mapping modules. Thus, we believe that, despite using a simplified state representation, a policy with a similar state representation could be realistically be trained and deployed in an AV stack.

Ego-Vehicle State

The observation about the ego-vehicle state contains the following information:

[4] https://leaderboard.carla.org/#evaluation-and-metrics

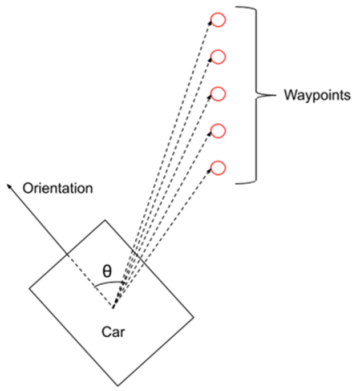

Figure 3.1: Visualization of η, the mean angular heading error between the AV to the next 5 waypoints

- $v_t \in [0, 1]$: Current speed of the ego-vehicle, scaled by the maximum allowed velocity of the ego-vehicle.
- $\varphi \in [-1, 1]$: Current steer angle of the vehicle, scaled by the maximum steer angle of the ego-vehicle. It is important to note that this is not simply equivalent to the commanded steer angle as CARLA simulates the dynamics of actuating the wheels, resulting in a delay in achieving the commanded steer angle.

Global Planner State

The global planner provides high-level route guidance about the path the ego-vehicle should follow. Specifically, the global planner computes a sequence of dense waypoints, spaced approximately 2 meters apart, that trace a route between the source and destination positions. The waypoints provide basic lane-changing information for entering intersections, but do not contain local planning information about handling the behavior of other actors. To incorporate the global planner information into our state space, we include the following features:

- $\eta \in [-\pi, \pi]$: Average heading error between the ego-vehicle heading and the next 5 waypoints, as shown in Figure 3.1.
- $d_{lat} \in [-1, 1]$: Lateral error from the global planner trajectory.

Other Dynamic Actor State

The state of the other nearby dynamic actors is high-dimensional, but critical to planning effective actions. We consider two simplified representations of these states. The first simple representation includes the distance and velocity of the nearest vehicle within 20 meters and in the same lane as the ego-vehicle.

- $d_{leading} \in [0, 1]$: Distance to nearest leading vehicle within 20 m in the same lane as the ego vehicle.
- $v_{leading} \in [0, 1]$: Speed of the leading vehicle.

We concatenate the obstacle observations for brevity as follows:

$$\overrightarrow{\mathbf{O}}_{simple} = [d_{leading}, v_{leading}]$$

While very simplistic, this representation is sufficient for simple scenarios involving single lane roads and other vehicles behaving predictably. Most scenarios in the NoCrash benchmarks can be solved with this representation. In order to scale to more challenging scenarios and other dynamic actors such as pedestrians, we need to consider a representation that contains information about actors beyond those directly in front of the ego-vehicle.

The second, expanded representation splits the area of around the ego-vehicle into 5 sections: front, front-left, front-right, back-left, and back-right. The representation includes the relative position and velocity vectors for the nearest actors, including bicyclists and pedestrians, for each region. Additionally, we increase the detection range in this representation to 45 meters. Concretely, the representation is defined as follows.

- $\overrightarrow{\mathbf{d}}^{\beta}_{\text{actor}} \in [-1, 1.5]^2$: Cartesian coordinates in ego-vehicle frame of the nearest vehicle in obstacle range in region defined by β, scaled by max detection range.
- $\overrightarrow{\mathbf{v}}^{\beta}_{\text{actor}} \in [-1, 1.5]^2$: Velocity in ego-vehicle frame of the nearest vehicle in obstacle range in region defined by β, scaled by max detection range.

The representation for each detection range is then defined as:

- $\overrightarrow{\mathbf{O}}_{\text{front}} = \left[\overrightarrow{\mathbf{d}}^{[-5.73,5.73]}_{\text{actor}}, \overrightarrow{\mathbf{v}}^{[-5.73,5.73]}_{\text{actor}}\right]$
- $\overrightarrow{\mathbf{O}}_{\text{front,right}} = \left[\overrightarrow{\mathbf{d}}^{[5.73,90]}_{\text{actor}}, \overrightarrow{\mathbf{v}}^{[5.73,90]}_{\text{actor}}\right]$
- $\overrightarrow{\mathbf{O}}_{\text{front,left}} = \left[\overrightarrow{\mathbf{d}}^{[-90,-5.73]}_{\text{actor}}, \overrightarrow{\mathbf{v}}^{[-90,-5.73]}_{\text{actor}}\right]$

- $\overrightarrow{\mathbf{O}}_{\text{back,right}} = \left[\overrightarrow{\mathbf{d}}_{\text{actor}}^{[90,180]}, \overrightarrow{\mathbf{v}}_{\text{actor}}^{[90,180]}\right]$
- $\overrightarrow{\mathbf{O}}_{\text{back,left}} = \left[\overrightarrow{\mathbf{d}}_{\text{actor}}^{[-180,-90]}, \overrightarrow{\mathbf{v}}_{\text{actor}}^{[-180,-90]}\right]$

By providing position and velocity information for actors in all directions, this observation space enables the motion planner to handle more challenging traffic scenarios such as merging and pedestrians crossing the road.

Traffic Control State

Finally, motion planning in urban settings requires the ego-vehicle to obey traffic lights and stop signs. Currently, we only consider traffic lights. To simplify the offline learning setup, we simply encode a red traffic light as an obstacle with zero velocity in front of the ego-vehicle.

Full observations

Combining these features, we consider two observation spaces, each with a different representation for dynamic actors. The first observation space, which we name Front Obstacle observation, is defined as:

$$[v_t, \varphi, \eta, d_{lat}, \overrightarrow{\mathbf{O}}_{simple}]^T$$

The second observation space, named 360 Obstacle observation, is defined as:

$$[v_t, \varphi, \eta, d_{lat}, \overrightarrow{\mathbf{O}}_{front}, \overrightarrow{\mathbf{O}}_{\text{front,right}}, \overrightarrow{\mathbf{O}}_{\text{front,left}}, \overrightarrow{\mathbf{O}}_{\text{back,right}}, \overrightarrow{\mathbf{O}}_{\text{back,left}}]^T$$

3.2.2 Action Space

We choose an action space similar to the one used in [1]. We assume access to a PID controller capable of providing brake and throttle commands to track a target velocity. The learned motion planning policy provides a target velocity v_{target} as input to the PID controller. The action space is biased slightly, such that a command of 0 maps to a positive velocity. To control the steering, the policy directly provides a steer angle ϕ_{target} that is tracked by the vehicle.

- $v_{target} \in [-1, 1]$
- $\phi_{target} \in [-1, 1]$

3.2.3 Reward and Termination Function

We consider the following reward function for the MDP.

$$\mathcal{R} = \alpha_v \mathcal{R}_v + \alpha_{lat} \mathcal{R}_{lat} + \alpha_{collision} \mathcal{R}_{collision} \tag{3.1}$$

$\mathcal{R}_v$ is a positive reward proportional to the current speed of the vehicle, $\mathcal{R}_{lat}$ is a negative reward proportional to the lateral trajectory error, and $\mathcal{R}_{collision}$ is a constant negative penalty for collisions. For all of the experiments presented in this work, all weights in the reward function are 1. In the future, we plan to explore adding terms that explicitly encourage smooth driving as well.

The episodes in the MDP terminate if any of the three following conditions are met.

- The ego-vehicle successfully reaches the end of the trajectory
- The ego-vehicle collides with another actor
- The ego-vehcile drifts too far from the center of the lane

Chapter 4

Offline Model-Based Reinforcement Learning

The objective of model-learning in MBRL is to learn a generative model of the state-transition distribution $p_\phi(s'|s,a)$. This model can then be used for online planning or policy optimization. As detailed in Section 2.1.3, the offline setting restricts this formulation by only providing access to a pre-collected dataset of experience, rather than providing access to the environment.

At a high level, our algorithm, drawing from ideas presented in Model-Based Policy Optimization (MBPO) [11], learns a dynamics model from the dataset. We can think of this learned dynamics model as a surrogate simulator and use it in that capacity to optimize a policy. Figure 4.1 presents a high level diagram of this procedure. In the next section, we explore the challenges in this algorithmic framework, and proceed to explore our solutions to them.

4.1 Why is this hard

Model-based policy optimization, presented in [11], demonstrates strong results by using a learned dynamics model in conjunction with a model-free RL algorithm to optimize a policy. At a high level, MBPO collects experience in the environment, fits a dynamics model to the data, optimizes the policy using the dynamics model as a simulator, and repeats this loop. Though this algorithm is designed to train a

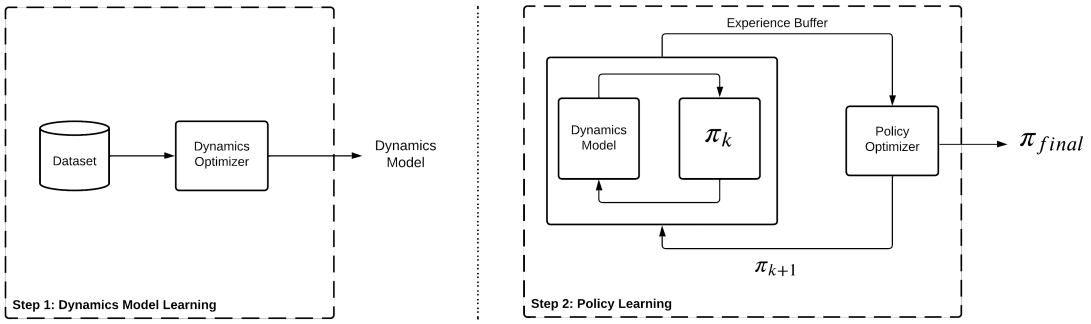

Figure 4.1: High-level overview of training procedure in Offline MBRL. We first learn a dynamics model of the environment and then use that as a simulator to optimize a policy.

policy online, it appears to be well-suited to run offline by using the pre-collected dataset to train the dynamics model and optimizing the policy without collecting additional data. However, MBPO tends to fail catastrophically when naively ported to the offline setting [22].

To understand the challenge with dynamics modeling in the offline setting, we present a toy MDP as shown in Figure 4.2. Figure 4.2(a), we see a dataset of trajectories collected by the data generation policy that is used to train the dynamics model. In Figure 4.2(b), we now see a new trajectory predicted by the dynamics model given a sequence of actions generated by the policy unseen in the dataset. Given our intuition about the environment, this would appear to be a reasonable trajectory to predict. However, the actual MDP has a large negative reward region in the region of the state space in which the dynamics model predicted the trajectory as shown in Figure 4.2(c).

In this case, the dynamics model incorrectly generalized to predict a trajectory that was not feasible. However, in the case that the environment did not contain this large negative region, the dynamics model actually correctly generalized and allowed the policy to plan a higher reward path than the demonstrated one. Given only the demonstrations in the dataset, it is impossible for the model to distinguish between these two cases.

When trained online, MBPO has two advantages over the offline setting. First, since data is collected with the current policy, the dynamics model only needs to

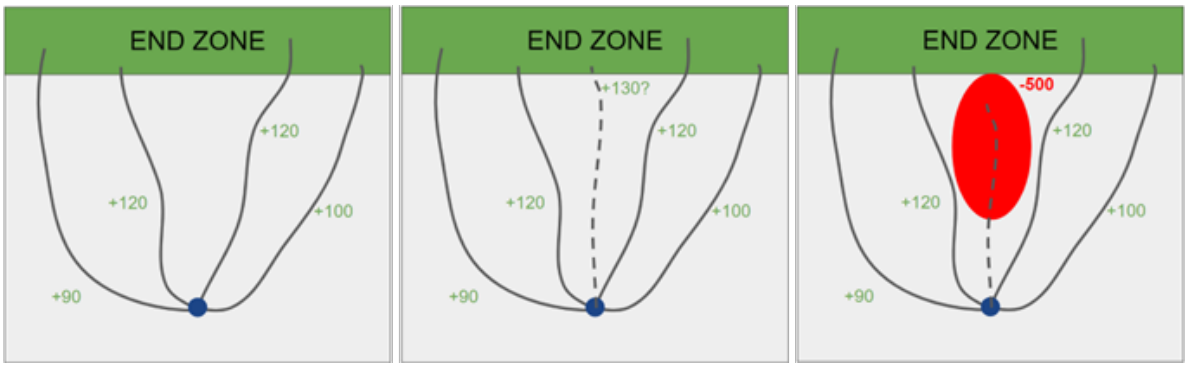

Figure 4.2: Illustrative Toy MDP to demonstrate challenges with Offline RL. (a) depicts the dataset use to train the policy in the toy MDP. (b) represents a trajectory planned by the policy in the dynamics model. In (c), we see that trajectory actually leads to a large negative reward, which dy namics model was unaware of due to the lack of data

locally model the transition dynamics in the state-action distribution induced by the policy. Assuming a limited modeling capacity, the dynamics model can be more accurate in the local region than an equivalent model that has a requirement to be globally accurate. The second advantage of the online setting is that the policy can collect counterfactual examples to correct dynamic modeling errors that do exist. Returning to the toy MDP, consider the case in which a policy learned to execute the trajectory predicted in Figure 4.2(c) given a model trained on the dataset in Figure 4.2(a). When the policy is re-deployed to collect new data in the environment, it will collect many trajectories interacting with the negative reward region, which will correct the belief of the dynamics model about that that region in the state-action space. This counterfactual data makes the algorithm robust to modeling errors. In the offline setting, we never have access to these counterfactual demonstrations during training, meaning that modeling errors can lead to catastrophic failures during test-time execution.

This problem we outline is often referred to as distributional shift [15]. To handle distributional shift, we have two options. The first option is to constrain the policy to stay near the data distribution demonstrated in the dataset. Naturally, this poses imitation learning algorithms as good options as they optimize the policy to be similar to the data generation policy. Both MOPO [22] and MOReL[12] attempt to improve performance over imitation learning by performing RL using the dynamics model while

penalizing the agent for regions of the state-action space in which the dynamics model is uncertain. Effectively, the weight of the penalization serves as a balance between pure imitation learning and pure RL. The choice of the penalization hyperparameter is problem dependent and often not obvious, even given prior knowledge about the environment.

The other option to handle distributional shift is to incorporate prior knowledge about the environment into the dynamics model to allow for safe generalization beyond the provided data distribution. This prior knowledge could be incorporated in the form of a known reward or termination function, or a closed-form model of the state-transition dynamics. For the toy MDP, a known reward function would allow the model to safely generalize in the safe regions of the MDP, while ensuring that the policy never learns to enter the low-reward region. The best method for incorporating prior knowledge is heavily problem dependent. In the case of driving, we have strong knowledge about the structure of the problem that we can exploit to construct a model that generalizes safely.

4.2 Dynamics Learning

Given the problem formulation of motion planning for AVs, we know that the environment is stochastic and partially-observable. Both of these characteristics make it challenging to build an accurate and stable dynamics model that directly predicts in the policy observation space. However, similar to the observation made by [4], we note that the environment can be decomposed into two parts: the dynamics of the ego-vehicle and the dynamics of the other actors on the road. Given the policy action and current state of the ego-vehicle, the ego-vehicle dynamics are deterministic. The dynamics of the other vehicles and pedestrians are stochastic as we cannot observe their intention or driving styles. To handle the different natures of these two portions of the environment dynamics, we choose to separately model the ego-vehicle dynamics and the dynamics of the other vehicles. In the following sections, we describe the ego-vehicle model and other vehicle model.

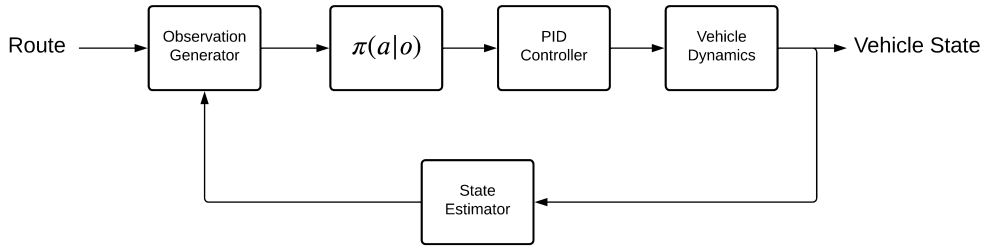

Figure 4.3: Block Diagram for Ego-Vehicle Closed Loop Control. The policy π sends controls to the PID controller, which actuates the vehicle. The observation encodes the new the state of the vehicle, which is estimated by the state estimator.

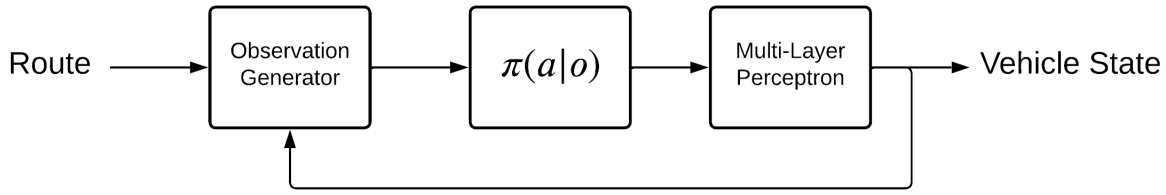

Figure 4.4: Block Diagram with a learned dynamics model. We see that the MLP models the dynamics of the PID controller, vehicle, and state estimator.

4.2.1 Ego-Vehicle Dynamics

A block diagram of the ego-vehicle planner and controller are presented in Figure 4.3. As apparent from the diagram, modeling the dynamics of the vehicle for the motion planner does not only involve modeling the dynamics of the vehicle itself, but also the dynamics of the motion controller and state estimator. While building a closed form model for these dynamics is possible, we opt to model the dynamics of the vehicle with a multi-layer perceptron (MLP). An MLP is able to fit non-linearities in the vehicle dynamics better than commonly used closed-form models such as bicycle models. Furthermore, MLPs will likely scale better to handle more challenging dynamics such as slippery road conditions without requiring extensive modeling work. A block diagram of our model is shown in Figure 4.4. The MLP input space contains the following features.

- Vehicle velocity at times t and $t-1$ (v_{t-1}, v_t)
- Vehicle steer angle at times t and $t-1$ (φ_{t-1}, φ_t)
- Actions at t and $t-1$ (a_{t-1}, a_t)

The dynamics model predicts the change in the SE2 pose of the vehicle as well as the change in speed and steer angle. With this data, we can auto-regressively roll out the dynamics model to predict the motion of the vehicle given a sequence of actions.

4.2.2 Other Actor Dynamics

Modeling the dynamics of the other actor is a more challenging task because we cannot observe their intention. A potential option could be to apply the latest techniques in the literature of motion forecasting to model these behaviors [8, 10, 17]. However, most motion forecasting models tend to diverge to unreasonable predictions over longer horizons (more than 10 seconds).

Instead, we opt for a more simplistic model in which we assume that the other actors are non-reactive. With this assumption, the trajectories of the other agents can simply be replayed from the dataset. This completely eliminates the need for any learned forecasting models while maintaining realism in the trajectories. While this assumption does limit the types of scenarios that the policy can learn to handle, we find that it is a great starting point for learning effective driving policies. This

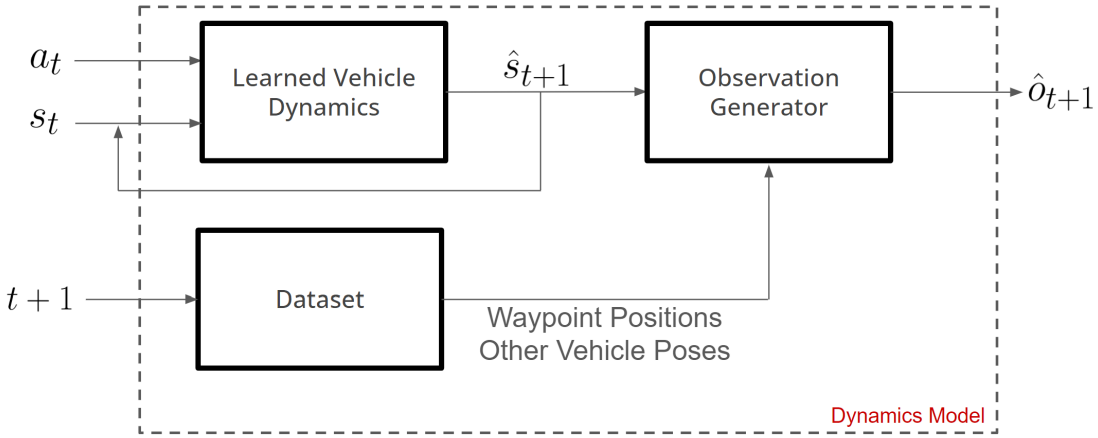

Figure 4.5: A diagram of the architecture of the full dynamics model. The learned vehicle dynamics autoregressively predicts the motion of the ego-vehicle while the motion of the other vehicles is replayed from the dataset. These features combined are used to compute the policy observation.

modeling decision is similar to the one made by World on Rails [4], so we refer to this assumption as the "World on Rails Assumption". A full diagram of the dynamics model is presented in Figure 4.5

4.2.3 Training Procedure

We collect two datasets in CARLA to train the ego-vehicle dynamics model. The first dataset contains 200,000 state transitions generated by uniformly sampling actions from the action space. The second dataset contains 200,000 state transitions generated by a heuristic autopilot policy with Gaussian noise injected into the actions. In total, this is equivalent to around 11 hours of driving demonstrations.

As presented in [12, 22], we opt to train an ensemble of deterministic dynamics models to predict the future state. The ensemble enables us to compute an estimate of modeling uncertainty and can provide robustness for policy optimization by allowing the policy to train against slightly different "guesses" for the environmental dynamics. Each of the dynamics models are trained on the entire dataset, but are randomly initialized and trained on different samplings of training batches. Each model is trained to minimize one-step prediction error using the Huber loss. We choose Huber loss to minimize the effects of outliers on model training.

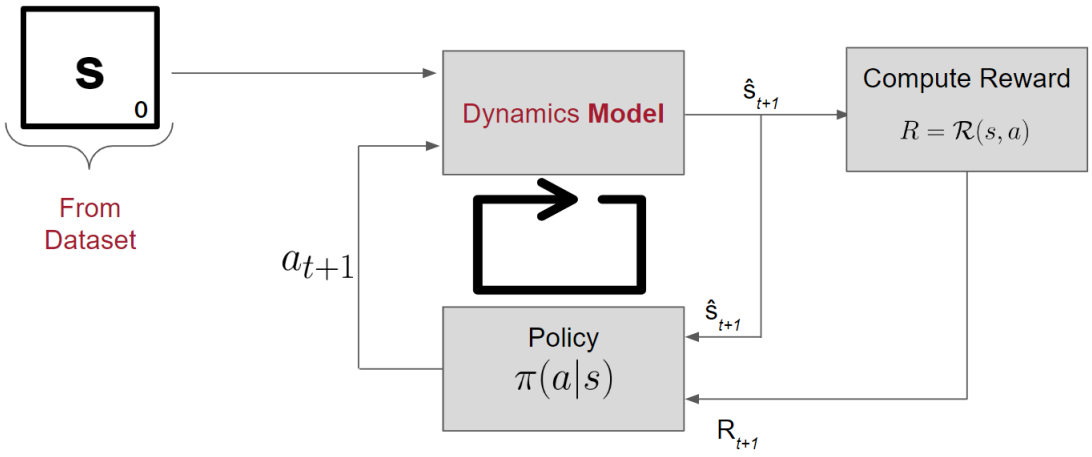

Figure 4.6: Policy optimization loop for offline MBRL training. The dynamics model is used as a simulator that a model-free policy optimizer can use to train the policy

$$\mathcal{L} = \begin{cases} 0.5 * (\hat{s}_{t+1} - \hat{s}_{t+1}) & \textit{if } |\hat{s}_{t+1} - \hat{s}_{t+1}| < \delta \\ \delta \left(|\hat{s}_{t+1} - \hat{s}_{t+1}| - 0.5 * \delta \right) & \textit{otherwise} \end{cases} \tag{4.1}$$

4.3 Policy Learning

We now present a procedure for policy optimization making use of the learned dynamics model and discuss the performance on benchmarks.

4.3.1 Algorithm

Figure 4.6 contains a diagram of the training loop for the policy, which closely resembles the optimization procedure followed for model-free algorithms. Since we are treating the dynamics model as a simulator, any model-free RL algorithm can be used to optimize the policy.

This optimization procedure is similar to those presented in MOPO and MOReL [12, 22] with some simplifications. Both algorithms use prediction discrepancy between the members of the dynamics ensemble as a measure of uncertainty of the dynamics model. MOPO [22] adds a penalty in the reward function proportional to the

magnitude of the uncertainty term, while MOReL [12] prematurely terminates episodes if the uncertainty exceeds a threshold. However, due to the strong performance of our dynamics model, we find that we can completely remove these penalties on model uncertainty with no adverse effect on performance. Furthermore, we find that the value function bootstrapping procedure with truncated rollouts presented in MOPO [22] to be detrimental to policy performance.

4.3.2 Training Details

We train the policy on a separate dataset from the ones used to train the dynamics model. For the NoCrash benchmarks, we collect a dataset of 200,000 interactions generated by a heuristic autopilot with no gaussian noise. For Leaderboard, we collect the dataset of 200,000 interactions using the autopilot presented in Transfuser [5]. We choose these specific autopilots due to their strong performance which generates long rollouts of training experience.

To fully understand why this is important, consider a poor policy that always collides with the leading vehicle. Since the MDP formulation includes terminations on collisions, the dataset does not contain trajectories that extend beyond these collisions. Since we cannot infer the behavior of the other agents beyond the collision, we assume that the episode terminates at the timestep at which the demonstration policy collided. Training a policy with this dataset would likely result in a policy that simply delays the collision slightly because this would result in a successful termination. With a strong performing autopilot, the dataset would contain trajectories beyond these interaction because the episodes would not prematurely terminate. This more closely resembles the datasets available for AV companies, as the trajectories include safety driver take-overs in cases where the autonomy stack fails in challenging sitations.

We use PPO [19] as our model-free policy optimization algorithm. The full list of hyperparameters can be found in the appendix.

4.3.3 Results

We present results on the NoCrash and Leaderboard benchmarks. We compare our algorithm to Transfuser [5], World on Rails (WOR) [4], and MP-PPO [9], which are some of the top performing algorithms on the Leaderboard benchmarks. For

all benchmarks, we report results for policies trained for 2 million steps. While we could periodically evaluate our policy on the benchmarks to select the best one, this would not be possible in the real driving setting without a strong off-policy evaluation method. Therefore, we select a policy at a fixed timestep.

NoCrash

We used the Front Obstacle observation space for the NoCrash benchmarks. Table 4.1 presents the quantitative results for the NoCrash benchmark. Our method nearly achieves a perfect score on the benchmark and is comparable in performance to the best baselines.

We see a significant performance improvement over WOR, which makes a similar assumption about replaying the behavior of other vehicles from the dataset. There are many possible variables that could explain the performance gap. A key difference could be the stronger performance of our dynamics model over the parametrized bicycle model that WOR uses, as discussed in section 4.2.1. A large difference in the dynamics predictions of the model and the real environment can cause the policy to take suboptimal actions in the real environment, similar to the issue of sim2real transfer that shows up when training in a simulator. Another important difference is that WOR explicitly runs a planner with a 10 second horizon at each state in the dataset to produce an action label for that state. This can lead to poor policy performance at test time as the state visitation distribution of the learned policy shifts from that of the demonstration policy. If the policy learns to select significantly different actions than the demonstration actions, it can visit states that were not present in the dataset, leading to unstable action predictions. Finally, WOR learns a policy that takes raw sensor data as observation instead of our low dimensional state representation, which can make it challenging for their policy to generalize. We plan to perform ablations on WOR in future works to identify the root cause of this performance difference.

Our policy performance is comparable to the performance of MP-PPO, which is the most similar benchmark. MP-PPO trains policies with the same MDP defintion as ours, but in a fully online setting. We can see here that the switch to the offline setting has not reduced the policy performance on the benchmark compared to the

Table 4.1: Quantative Performance Results on the NoCrash benchmarks for Self-Play. Each value represents the percentage of successful episodes out of 25 routes, averaged over 3 seeds and 3 repeats of each route. Autopilot refers to the data generation heuristic policy without noise injected into the actions. Our algorithm achieves a near perfect score on the benchmark, matching the performance of the state-of-the-art online RL algorithm MP-PPO.

	Autopilot		WOR		MP-PPO		Ours	
	Train	Test	Train	Test	Train	Test	Train	Test
Empty	100	100	98	94	100	99	100	99
Regular	100	100	100	89	99	99	100	99
Dense	100	88	96	74	100	96	100	96

online setting.

Leaderboard

Next, we evluate the performance of our algorithm on the leaderboard benchmark. we train our policy with the 360 Obstacle observation to enable the policy to reason about more challenging scenarios such as lane changing. The results are presented in Table 4.2. We compare the performance of our policy against Transfuser [5] which achieves state-of-the-art performance on the official held-out `testing` routes, and MP-PPO which achieves similar performance to Transfuser on the `validation` routes. As discussed in Section 3.1.2, we only report performance on the `validation` routes as we make use of privileged simulator information.

We see that our performance on leaderboard is significantly worse than that of Transfuser and MP-PPO. Our policies achieve a relatively high route completion, but incur large penalties due to collisions and red light infractions. This large gap in performance may be a limitation caused by the World on Rails assumption. We can view this assumption as a regularizer that promotes the policy to stay near the state-visitation distribution of the demonstration policy. When the policy drifts too far from the demonstration distribution during training, it encounters cases in which the other agents collide with the ego-vehicle from the side or rear, even though the driving policy is effective. Since leaderboard involves longer horizon episodes with more agents, this effect is exacerbated on this benchmark over NoCrash. Next, we discuss extensions to this algorithm to move beyond this assumption.

Table 4.2: Leaderboard Evaluation scores on the validation routes. "DS" stands for Driving Score, the aggregate score computed by the Leaderboard benchmark. "RC" stands for perecentage of route completion. "Collis." stands for collisions (Unit: # / kilometer). "Viol." combines red light, stop sign, and off-road violations (Unit: # / kilometer). "Dev." stands for route deviations, which are significant deviations from the global plan (Unit: # / kilometer). "Timeouts" combines the number of times the episode terminates due to the agent being static (Unit: # / kilometer).

	DS	RC	Collis.	Viol.	Dev.	Timeouts
Transfuser	83.95	99.63	0.04	0.08	0	0.01
MPPPO	80.64	99.27	0.06	0.05	0	0.03
Ours	28.62	86.09	31.45	18.24	0	12.04

Figure 4.7: Example of a scenario requiring the vehicle to negotiate right-of-way with another vehicle. The ego-vehicle (red car) needs to merge into the lane with moving traffic (green cars). With the World on Rails assumption, the green cars do not slow down as the ego-vehicle enters the lane, resulting in a collision.

4.4 Self-Play

While training with the World on Rails assumption demonstrated strong performance on the simple benchmarks, the algorithm fails to produce strong results on the more challenging Leaderboard benchmark. This failure highlights a key limitation of the World on Rails assumption: the behavior of the other agents does not change as the behavior of the ego-vehicle changes. This limitation causes instability in training for leaderboard, but also has implications for the types of policies that can be learned.

Consider the scenario presented in Figure 4.7, in which the ego-vehicle is attempting to merge into a lane with moving traffic. Deciding whether to yield for the vehicle

entering the intersection is a challenging interaction, that is dependent on the intent of both the AV and the other actor. Aggressive behavior from the ego-vehicle could signal to the other actor to yield and vice versa. Regardless of the outcome, in the real world, both agents must predict the behavior of the other agent to plan actions. With the World on Rails assumption, the AV only interacts with non-reactive agents during training, which would not slow down when the AV enters the lane. Therefore, the policy would never be able to make forward progress on the route.

To handle these more challenging scenarios, we must move beyond the World on Rails assumption. As discussed previously, building motion forecasting models that are stable over long horizons and that model the true distribution of behavior is quite challenging. However, instead of modeling the end-to-end behavior of the other vehicles, we can further decompose the other vehicle dynamics to build a stable dynamics model. While the intent and driving style of the other vehicles is not observable and is hard to model, the dynamics of their vehicle are deterministic and similar to the dynamics of the ego-vehicle. Therefore, given we know the action that the other agents take, the behavior of the other actor is deterministic. If we assume that all of the vehicles have the same dynamics as ours, we can use the ego-vehicle dynamics model to predict the motion of all cars.

To generate actions for the other vehicles, we can deploy any policy on the other vehicles such as the heuristic autopilot, the current ego-vehicle policy, or past policy checkpoints. This allows us to seed reactive policies on all of the vehicles, thereby removing the World on Rails Assumption. This setup also creates a self-play, multi-agent training environment, in which we can collect data from all of the agents to train a single policy or set of policies. Finally, by using suboptimal policies such as intermediate checkpoints, we can generate challenging long-tail scenarios for training. In the following sections, we outline our self-play algorithm, demonstrate its performance on the toy merging example presented in Figure 4.7, and present its results on NoCrash.

4.4.1 Algorithm

A diagram of the self-play training loop can be seen in Figure 4.8. This loop closely resembles the training loop for offline MBRL with the World on Rails assumption,

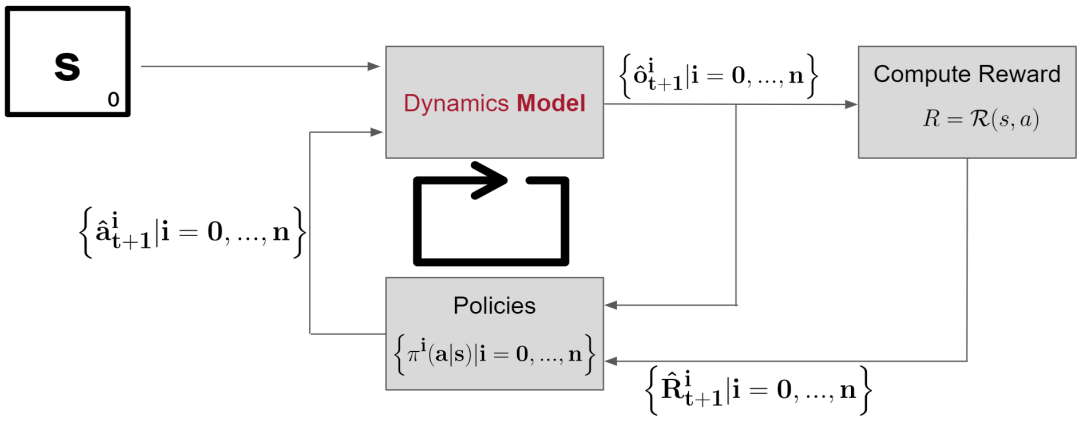

Figure 4.8: A digram of the training loop we use to perform self-play. Instead of predicting the behavior of just the ego-agent, we predict the behavior for all actors in the environment using the dynamics. The policy optimizer can make use of all of the data from these models to train the policy.

except that we collect data from a set of policies instead of a single policy.

Since we do not have access to the full observations from the other vehicles, we have to make assumptions about their intention to generate valid observations. The key element of the observation space we are missing for other vehicles is the global plan they are following. To generate an estimate for this plan, we make the assumption that the other vehicles are effectively tracking their global plan throughout their trajectory. With this assumption, we can simply sparsify their trajectory to generate a global plan. Along with the initial speed and pose of the other vehicles, we have enough information to compute the observations.

To perform policy optimization, we make use of parallel RL algorithms proposed in [9]. Particularly, we use MP-SAC, which enables us to use the data we collect from all of the vehicles in the environment to train the policy as well as run multiple training environments in parallel. With this algorithm, we take advantage of the data generated from all of the agents in the environment, leading to a training speed increase.

4.4.2 Toy Problem

To demonstrate how the self-play setup allows the policy to learn driving behaviors beyond those that can be learned with the World on Rails assumption, we train the policy for the toy problem in Figure 4.7, in which the ego-vehicle is merging into a lane with flowing traffic. In the dataset, the vehicles are travelling at 20 km/hr along the lane spaced 20 meters apart. When training with the World on Rails assumption, the trajectories of these vehicles are simply replayed from the dataset. To train in the self-play setting, we deploy the heuristic autopilot policy on each of these vehicles and the optimize the neural network policy on the ego-vehicle. At test time, both agents are deployed in an environment in which the other agents are reactive.

The policy trained with the World on Rails assumptions completely fails to enter the flowing traffic and instead departs the lane before reaching the other agents. This is likely because, during training, any attempt to enter the flowing traffic would cause a collision as the rear car does not slow down for the ego-vehicle. On the other hand, the policy trained in the self-play setup learns to enter the traffic successfully for every evaluation run. In contrast to the World on Rails setting, the policy was able to learn to negotiate the intersection with the other agents since they were reactive in the training environment. While this is a very simplified example of the self-play setting, this demonstrates the importance of self-play in scenarios requiring negotiation with other agents, which constitute a large portion of challenging long-tail scenarios.

4.4.3 NoCrash

Next, we evaluate the performance of the self-play algorithm in NoCrash. The policy training dataset is the same as the dataset used for the World on Rails setting, presented in Section 4.3.2. Each episode in the dataset contains trajectories for 100 agents and the ego-vehicle. At each episode, we randomly deploy the current neural network policy on 40 agents, which are all used to collect data. The remaining 61 agents use the heuristic autopilot policy to take actions, which are not added to the experience buffer. The heuristic autopilots ground the behavior of the learned policy by providing demonstrations of normal driving behaviors. If all of the agents use the learned policy instead, the policy could learn driving behaviors that diverge from strategies that would work at test time, an effect discussed in [21]. Finally, utilizing

Table 4.3: Quantative Performance Results on the NoCrash benchmarks for Self-Play. Each value represents the percentage of successful episodes out of 25 routes, averaged over 3 seeds and 3 repeats of each route. Autopilot refers to the data generation heuristic policy without noise injected into the actions. Self-Play achieves a perfect score on Empty and Regular, but performs worse than state-of-the-art on Dense.

	Autopilot		WOR		MP-PPO		Ours		Self-Play (Ours)	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Empty	100	100	98	94	100	99	100	99	100	100
Regular	100	100	100	89	99	99	100	99	100	100
Dense	100	88	96	74	100	96	100	91	92	72

the parallelism of MP-SAC, we also run 5 environments in parallel, for a total of 200 agents collecting data in parallel.

Table 4.3 presents the results of our self-play algorithm on the NoCrash benchmark. We see that self-play achieves a perfect score in NoCrash Empty and Regular, but has a higher failure rate than state-of-the-art on NoCrash Dense. This reduced score is due to increased instability in training caused by the multi-agent training environment. Particularly, this raises the issue of adverserial agents in the environment. During training, agents can receive negative rewards for collisions caused by bad actions taken by other actors. This increases the variance of the gradients, resulting in a more unstable optimization procedure. Handling this issue is particularly challenging in the driving problem as the problem not fully cooperative nor fully competitive. In most cases, all actors on the road behave cooperatively, but there are many instances of competitive behavior as well. If we train the policy assuming all drivers are cooperative, the policy will never learn to handle the challenging edge cases in which another agent is acting agressively. However, a fully competitive environment will likely result in a policy that never learns to drive as other agents are constantly crashing into the ego-vehicle. It is important to reach a realistic balance between these two extremes to train a successful policy.

Chapter 5

Conclusions

In this thesis, we study the challenges of offline reinforcement learning for autonomous vehicle motion planning. We propose two model-based reinforcement learning algorithms that make use of dynamics models that exploit the structure of the driving problem to generalize successfully. We demonstrate that these methods can match state-of-the-art performance on the simpler CARLA benchmarks and explore the limitations on their performance on the more challenging benchmarks.

While these methods have not yet reached state-of-the-art performance, they have potential to be valuable algorithms for learning motion planners in real autonomous vehicle stacks. Firstly, their ability to make use of pre-existing driving logs allows them to take advantage of the datasets of driving experience AV companies maintain. Additionally, the self-play training environment has the potential to massively parellelize training to take full advantage of the large scale of the available data.

There are many avenues for future works that are important to improve the performance of these algorithms and make them viable to be deployed at AV companies. First of all, improving self-play likely requires investigation into better algorithms to handle the multi-agent training setup. In this work, the policy optimization ignores the multi-agent nature of the problem, leading to issues such as incorrect credit assignment in collisions. Applying advances in the work of multi-agent RL could help significantly stabilize the performance of this training setup. Furthermore, while we choose SAC to allow for off-policy training, SAC tends to perform poorly when trained with off-policy data, limiting our ability to make use of experience

from heteregeneous agents. Work such as AWAC [16] address this issue to allow the policy to learn effectively from off-policy data. It would be valuable to integrate such off-policy algorithms to improve the sample-efficiency of the self-play algorithm.

For self-play to scale to generate challenging long-tail scenarios requires investigation into the balance between cooperative and adverserial driving policies. Currently, we only consider optimizing a single policy during training. However, the multi-agent setup could enable training a heterogeneous set of driving policies with varying objectives or using behavior cloning to generate policies with different driving styles. AV companies can also deploy and collect data from various iterations of their existing motion planners to produce further diverse training data. To better control the difficulty of the driving scenarios, exploring curriculum learning for scenario generation could be valuable. A curriculum learning setup could enable the training setup to generate increasingly difficult long-tail scenarios, while not making them impossibly challenging.

A key limitation we have not explored in detail in this work is the challenge of occlusions in the driving datasets. Our dataset assumes ground-truth access to the state of all vehicles, regardless of whether or not they are occluded. In a real driving dataset, we do not have access to the full trajectories of all vehicles as many will be occluded through the course of the episode. Recently, a new benchmark called Nocturne [20] has been released to explore exactly this challenge. Designing a multi-agent training environment that reasons about the state of actors through occlusions will be critical to deploying such a training setup in a real AV stack.

Another key issue that this work does not address is that of off-policy evaluation, which is evaluating the performance of the policy before deploying it in the test environment. While training rewards are correlated with test time performance, we often find policies with similar rewards that perform very differently at test time. Without an effective method to evaluate the policies before deploying them at test time, it is challenging to tune hyperparameters and to select the best policy from training.

We can also reduce the importance of off-policy evaluation, however, if we relax the requirement of training policies strictly offline. It is unrealistic to assume that AV companies would deploy policies trained completely offline directly into production. The more likely situation is that after a policy is trained offline, the policy will be

deployed for evaluation to collect additional data which can be used for fine-tuning. While this data can simply be used by adding it to the training dataset and continuing training, a more intelligent strategy for utilizing that data could make the limited online experience more valuable. Intelligent exploration algorithms or methods to evaluate a set of policies could be valuable to maximize the utility of the on-policy data.

Finally, many of the concepts we have explored in this work try to sidestep the problem of motion forecasting through assumptions about the dynamics of the other vehicles. In the self-play training, we have loosely cast the motion forecasting problem into a planning problem. Indeed, motion planning and forecasting are very similar problems with slightly different objectives and different observations. It would be valuable to explore this connection between these two separate areas of research more deeply and to share successful algorithmic decisions between them to further improve performance.

Appendix A

Hyperparmeters

Table A.1: Hyperparameters for Dynamics Ensemble Training

Notation	Description	Value
`n_models`	Number of Ensemble Members	5
`lr`	Learning rate	0.001
`n_layers`	Number of layers	4
`n_neurons`	Number of neurons per layer	1024
`drop_prob`	Dropout probability	0.3

Table A.2: Hyperparameters for PPO with World on Rails Assumption

Notation	Description	Value
`lr`	Learning rate	0.0003
ϵ	Clipping Range	0.2
δ_{clip}	Maximum Gradient Norm	0.5
γ	Discount Factor	0.99
`n_steps`	Steps between policy updates	2048
`n_epochs`	Training epochs per update	10
`batch_size`	Training batch size	64

Table A.3: Hyperparameters for MP-SAC

Notation	Description	Value
`lr`	Learning rate	0.0003
`buffer_size`	Replay Buffer Size	1000000
`train_freq`	Steps between training updates	1
`gradient_steps`	Gradient steps per update	1
γ	Discount Factor	0.99

Appendix B

Ablation Studies

With many algorithmic changes from similar algorithms such as MOPO [22] and MOReL [12], we present ablation studies to understand how these algorithmic choices affect the performance of the learned policy.

Uncertainty Estimation with Ensembles

MOPO and MOReL both make use of prediction disagreement between members of the dynamics ensemble as an estimate of model uncertainty. While we use a model ensemble in our work, we do not use the uncertainty estimate to modify the reward as in MOPO [22] or the termination function as in MOReL [12]. For the first ablation, we add this uncertainty estimate back into the model as a term in the reward function according to the procedure defined by MOPO. We consider 3 weights of this penalty and evaluate the performance of the learned policies on NoCrash. The results are presented in Figure B.1. We see that for $\alpha = 1$, the policy performance is slightly better than the performance without the penalty. However, increasing α further leads to a degradation of performance. While a well-tuned value for this parameter may improve policy performance, it is challenging to tune this parameter in practice as it requires the ability to evaluate the policy in the environment. Therefore, in the interest of reducing hyperparameters, we choose to omit the uncertainty estimate from the reward function in our algorithm.

Additionally, we consider removing the ensemble completely and instead optimize

Table B.1: Performance comparison on the NoCrash benchmark for various ablation studies. Numbers are percentages of successful episodes for 3 seeds for 3 runs on the benchmark. α is the weight of the uncertainty term in the reward function. Single Model indicates a policy trained using a single dynamics model, instead of an ensemble.

	Ours + Uncertainty Penalization						Single Model	
	$\alpha = 1$		$\alpha = 5$		$\alpha = 25$			
	Train	Test	Train	Test	Train	Test	Train	Test
Empty	100	100	100	98	27	16	100	100
Regular	100	100	100	99	24	6	100	100
Dense	100	96	100	93	10	4	100	96

a policy using only a single dynamics model. The results are presented in Table B.1. Unintuitively, we see no reduction in performance training with only one model, suggesting that the model predictions are quite accurate. However, without the regularizing effect of training with multiple models, it is possible that the policy would overfit if trained for longer. We plan to study this effect in future works.

Bibliography

[1] Tanmay Agarwal, Hitesh Arora, and Jeff Schneider. Affordance-based reinforcement learning for urban driving. January 2021. 2.2.1, 3.2, 3.2.2

[2] Nolan Bard, Jakob N Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G Bellemare, and Michael Bowling. The hanabi challenge: A new frontier for AI research. February 2019. 2.2.3

[3] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. December 2019. 2.2.1

[4] Dian Chen, Vladlen Koltun, and Philipp Krähenbühl. Learning to drive from a world on rails. May 2021. 2.2.1, 4.2, 4.2.2, 4.3.3

[5] Kashyap Chitta, Aditya Prakash, Bernhard Jaeger, Zehao Yu, Katrin Renz, and Andreas Geiger. TransFuser: Imitation with Transformer-Based sensor fusion for autonomous driving. May 2022. 2.2.1, 4.3.2, 4.3.3, 4.3.3

[6] Felipe Codevilla, Eder Santana, Antonio Lopez, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9328–9337, 2019. doi: 10.1109/ICCV.2019.00942. 3.1.2

[7] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 1–16. PMLR, 13–15 Nov 2017. URL https://proceedings.mlr.press/v78/dosovitskiy17a.html. 2.2.1, 3.1.1

[8] Junru Gu, Chen Sun, and Hang Zhao. DenseTNT: End-to-end trajectory prediction from dense goal sets. August 2021. 4.2.2

[9] Zhe Huang. Distributed reinforcement learning for autonomous driving. Master's thesis, Carnegie Mellon University Pittsburgh, PA, 2022. 2.2.1, 3.1.2, 4.3.3, 4.4.1

[10] Zhiyu Huang, Xiaoyu Mo, and Chen Lv. ReCoAt: A deep learning-based framework for Multi-Modal motion prediction in autonomous driving application. July 2022. 4.2.2

[11] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-Based policy optimization. June 2019. 2.2.2, 4, 4.1

[12] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. MOReL : Model-Based offline reinforcement learning. May 2020. 2.2.2, 4.1, 4.2.3, 4.3.1, B, B

[13] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit Q-Learning. October 2021. 2.2.2

[14] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-Learning for offline reinforcement learning. June 2020. 2.2.2

[15] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. May 2020. 2.1.2, 4.1

[16] Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. AWAC: Accelerating online reinforcement learning with offline datasets. June 2020. 5

[17] Jiquan Ngiam, Benjamin Caine, Vijay Vasudevan, Zhengdong Zhang, Hao-Tien Lewis Chiang, Jeffrey Ling, Rebecca Roelofs, Alex Bewley, Chenxi Liu, Ashish Venugopal, David Weiss, Ben Sapp, Zhifeng Chen, and Jonathon Shlens. Scene transformer: A unified architecture for predicting multiple agent trajectories. June 2021. 4.2.2

[18] OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P d Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. December 2019. 2.2.3

[19] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. July 2017. 2.2.1, 4.3.2

[20] Eugene Vinitsky, Nathan Lichtlé, Xiaomeng Yang, Brandon Amos, and Jakob Foerster. Nocturne: a scalable driving benchmark for bringing multi-agent learning one step closer to the real world. June 2022. 5

[21] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja

Huang, Laurent Sifre, Trevor Cai, John P Agapiou, Max Jaderberg, Alexander S Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, November 2019. 2.2.3, 4.4.3

[22] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. MOPO: Model-based offline policy optimization. May 2020. 2.2.2, 4.1, 4.1, 4.2.3, 4.3.1, B, B