Offline Learning for Stochastic Multi-Agent Planning in Autonomous Driving

Adam Villaflor

CMU-RI-TR-24-13

April 2024

School of Computer Science
The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

ABSTRACT

Fully autonomous vehicles have the potential to greatly reduce vehicular accidents and revolutionize how people travel and how we transport goods. Many of the major challenges for autonomous driving systems emerge from the numerous traffic situations that require complex interactions with other agents. For the foreseeable future, autonomous vehicles will have to share the road with human drivers and pedestrians, and thus cannot rely on centralized communication to address these interactive scenarios. Therefore, autonomous driving systems need to be able to negotiate and respond to unknown agents that exhibit uncertain behavior. To tackle these problems, most commercial autonomous driving stacks use a modular approach that splits perception, agent forecasting, and planning into separately engineered modules. However, fully separating prediction and planning makes it difficult to reason how other vehicles will respond to the planned trajectory for the controlled ego-vehicle. So to maintain safety, many modular approaches have to be overly conservative when interacting with other agents. Ideally, we want autonomous vehicles to drive in a natural and confident manner, while still maintaining safety.

Thus, in this thesis, we will explore how we can use deep learning and offline reinforcement learning to perform joint prediction and planning in highly interactive and stochastic multi-agent scenarios in autonomous driving. First, we discuss our work in using deep learning for joint prediction and closed-loop planning in an offline reinforcement learning (RL) framework (Chapter 2). Second, we discuss our work that directly tackles the difficulties of using learned models to do planning in stochastic multimodal settings (Chapter 3). Third, we discuss how we can scale to more complicated multi-agent driving scenarios like merging in dense traffic by using a Transformer-based traffic forecasting model as our world model (Chapter 4). Finally, we discuss how we can draw from offline model-based RL to learn a high-level policy that selects over a discrete set of pre-trained driving skills to perform effective control without additional online planning (Chapter 5).

Acknowledgments

These last six years have been some of the most difficult but also some of the most rewarding years of my life, and I am grateful for all the people that have helped me on this journey. First, I would like to thank my advisors, Professor John Dolan and Professor Jeff Schneider, for their mentorship and guidance. In addition, I would like to thank my committee members, Professor David Held and Professor Philipp Krähenbühl, for their time and expertise. I would like to thank my undergraduate research mentors, Professor Sergey Levine, Professor Pieter Abbeel, and Gregory Kahn, for inspiring me to pursue academic research.

I would like to thank my collaborators — Brian Yang, Ian Char, Viraj Mehta, Christoph Killing, Zhe Huang, Swapnil Pande, and Chloé Su — without whom this research would not be possible. I would also like to thank everyone in the CMU Robotics Institute and the Auton Lab for creating a collaborative environment conducive for this work. In particular, thank you to Youngseog Chung, Yeeho Song, Conor Igoe, Ben Freed, and Tejus Gupta for all the thoughtful and engaging discussions over the years.

I would like to thank the CMU Ultimate community for welcoming me in when I first moved to Pittsburgh. I would also like to thank Matthew Mosso, my best friend and roommate, for helping make this city feel like home. Thank you to my Spikeball family — David, Michelle, Vicki, Calvin, Eric, Art, Tiffany, Alice, Danji, Charlotte, and Nikhil — and the Catan squad — Mary Cundiff, Dooplis, Vanjie, and Daisy — for providing me an outlet for fun, even on the cloudiest Pittsburgh day. Thank you to my friends back home — Jarren Suzuki, Evan Martin, Omar Agha and Joshua Jokom — for your continued transcontinental friendship.

Thank you to my parents, Cindy and Rom Villaflor, for your endless love and support, and for always having my back. Lastly, thank you to my partner Alexandra Yoon-Hendricks for always supporting me and encouraging me even when an entire country separated us. I am eternally grateful for your love and patience over these last 6 years.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

Imagine a situation where a vehicle is exiting a crowded parking lot after a football game. Most drivers know that in bumper-to-bumper traffic, they should behave proactively by cautiously asserting themselves to make space for themselves and progress forward. While this is intuitive for human drivers, these highly interaction-dense scenarios are a major challenge for current autonomous vehicles. In these situations, we require a robust model of how other agents might behave, and crucially, how they might dynamically respond to our own actions.

However, many autonomous driving systems separate prediction of the surrounding agents and motion planning for the ego-vehicle into two distinct processes. Separating these modules limits the ego-vehicle's potential to anticipate how other agents will react to its own actions and how it could dynamically adapt to different behaviors from surrounding traffic. Thus, these traditional methods that separate prediction and planning can easily lead to suboptimal plans in interactive scenarios, like exiting a parking lot or merging on a highway, when they do not account for these dynamic mutual interactions. To avoid these issues and unify prediction and planning, autonomous vehicles need to predict how surrounding agents will interact with the ego-vehicle's potential actions, and adjust the ego-vehicle's plan accordingly in a closed-loop manner. Making these interactive predictions is particularly challenging due to the complex multimodal stochasticity of pedestrians and traffic. Other agents' trajectories often strongly depend on information that is not directly available to the ego-vehicle, such as their intended goals or driving styles. Depending on the given traffic situation, the same ego-vehicle trajectory could be met with substantially different responses from the surrounding agents. For example, when merging onto a highway, vehicles in the target lane could yield to the ego-vehicle, speed up to make room for the ego-vehicle, or ignore the ego-vehicle, depending on both their driving style and the ego-vehicle's actions. In this thesis, we study how we can use deep learned models to address these issues and perform joint prediction and planning in these complex stochastic environments.

1.1 Related Work

Algorithm	Mutually Reactive	Probabilistic Planning	Enumerates Modes	Multi-Agent Forecasting
Symphony [50]	✓	✗	✗	✓
hoplan [47]	✗	✓	✗	✓
GameFormer [49]	✗	✗	✗	✓
PDM-Hybrid [29]	✗	✗	✓	✓
Lookout [24]	✗	✓	✓	✓
MFP [106]	✓	✗	✓	✓
CfO [92]	✓	✓	✗	✓
MB2PO (Ours)	✓	✗	✗	✗
SPLT (Ours)	✓	✓	✓	✗
P2DBM (Ours)	✓	✓	✓	✓
HOLOS (Ours)	✓	✓	✓	✓

Table 1.1: Comparison of how different algorithms use forecasting for online motion planning or offline policy learning through simulation. **Mutually Reactive** refers to forecasting in an auto-regressive manner where all the traffic agents and ego-agent continually react to each other in a closed-loop. **Probabilistic Planning** indicates whether the planned ego-agent behavior is properly evaluated with respect to the stochastic distribution of potential responses from the environment. **Enumerates Modes** indicates whether planning considers all the distinct behavior modes of the learned distribution for the ego-vehicle. **Multi-Agent Forecasting** indicates whether the dynamics model used for planning represents the other vehicles and pedestrians in the scene as a multi-agent system instead of a unified stochastic environment. We find that our work is the only one that ticks all of these boxes.

In order for autonomous vehicles to be both safe and proactive when interacting with unknown stochastic agents, they need to be able to anticipate the various ways other agents could react to their own actions and how they could adapt to differing behaviors from surrounding traffic. In this thesis, we explore 4 different characteristics of how prediction is used with planning that we believe should facilitate safe yet proactive driving in real interactive scenarios:

1. **Mutually Reactive.** Plans that the environment will continually react to the ego-vehicle, and that the ego-vehicle can continually react to the environment in closed-loop.

2. **Probabilistic Planning.** Expects the environment response to be stochastic, and plans appropriately over this uncertainty.

3. **Enumerates Modes.** Explicitly considers all the different important modes of behavior available to the ego-vehicle during planning.

4. **Multi-Agent Forecasting.** Takes advantage of the inherent structure in autonomous driving and explicitly represent the environment as a multi-agent system with dynamic agents and road objects in order to facilitate better predictions.

As outlined in Table 1.1, only our work has all of the 4 previously mentioned characteristics. Most prior approaches that incorporate forecasting into planning, either directly online or indirectly through offline policy learning, do not perform mutually reactive and probabilistic planning where they anticipate potential stochastic interactions between the ego-vehicle and surrounding traffic. Many of these approaches [47; 49; 29; 24] instead use open-loop predictions for the ego-vehicle and the surrounding traffic and do not directly consider the stochastic joint interactions between all the agents in the future. Therefore, they cannot anticipate the future mutual reactions between the ego-vehicle and the surrounding agents in the scene, which can lead to suboptimal behavior like the "frozen robot problem." In contrast, approaches that do jointly model the interactions between the ego-vehicle and the other agents in the scene [50; 106] in a closed-loop manner often do not properly evaluate the ego-behavior with respect to the stochastic distribution of potential responses from other agents in the scene. Instead, approaches like [50] search for one promising joint scene outcome, which can lead to overly optimistic behavior, as we explore in Chapter 3.

While CfO [92] does do mutually reactive and probabilistic planning, it does not explicitly enumerate the learned behavior modes during planning and it is only evaluated in scenarios with one other interacting agent. Many recent works in deep motion forecasting with real-world driving data [76; 38; 106; 49; 77] output GMMs or other similar distribution models with explicit and discrete multimodal outputs. One of the main benefits of representing these multimodal distributions this way is that it makes it trivial to enumerate the important behavior modes of the learned distribution even if the distributions of these modes are imbalanced. In Chapter 4, we demonstrate how these type of explicit multimodal models can facilitate efficient planning and lead to superior performance. Finally, if we want to use learned multi-agent aware models for motion planning in real-world autonomous driving, it is important that we use architectures that can actually scale to large and diverse real-world datasets like graphical neural networks [93; 24; 25], Transformers [112; 38; 76; 77], or similar architectures [111].

1.2 Contributions

In this thesis, we discuss how we start with offline learning for robotic control in a simple deterministic setting, and how we iteratively build on our work until we conclude with tackling learning for motion planning in complex autonomous driving scenarios with real-world driving data.

In Chapter 2, we discuss our work in using deep learning for joint prediction and closed-loop planning in an offline reinforcement learning (RL) framework. This work explores the benefits of combining uncertainty-aware model-based RL and behavior-regularized model-free RL in a unified model-based policy optimization (MBPO) [51] framework. We evaluate our method in the simpler robotic locomotion setting where there is no multimodal stochasticity. Even in this simpler setting, this work demonstrates the importance of considering many different potential interactions between the agent and the environment during the learning and decision making process, and how this can be distilled into a policy.

In Chapter 3, we discuss our work that directly tackles the difficulties of planning in stochastic multimodal settings. Even approaches that jointly perform prediction and planning by forecasting potential futures for the entire scene can produce biased optimistic solutions by not disentangling the effects of the agent's actions and the multimodal stochasticity in the scene. Specifically, approaches that search for the single most likely or best potential joint scene outcome can lead to dangerously optimistic behavior because the surrounding environment could potentially respond in an entirely different way to the same agent actions. In this work, we propose a method that addresses this optimism bias by explicitly disentangling learned policy and world models, which allows us at test time to search for policies that are robust to multiple possible futures in the environment.

In Chapter 4, we discuss how we can scale to more complicated multi-agent driving scenarios like merging in dense traffic by using a Transformer-based [112] traffic forecasting model as our world model. By using Transformer neural networks with a vector-based representation for all the relevant entities in the scene, such as vehicles, pedestrians, traffic lights, and map points, we can learn a model that makes more accurate marginal predictions for all the dynamic agents in the scene. In particular, we use an architecture that explicitly predicts a multimodal trajectory distribution for each agent by conditioning on a discrete set of distinct learned anchor embeddings to predict a variety of potential behaviors. These specific design choices for the model allow us to efficiently plan online by evaluating many distinct behaviors for the ego-vehicle with respect to a variety of potential responses from all the other agents in the scene. We find that our approach of using these powerful forecasting models with fully reactive closed-loop planning outperforms our baselines on a suite of challenging CARLA scenarios in dense traffic, while avoiding the “frozen robot problem.”

In Chapter 5, we discuss how we can use ideas from offline model-based RL to learn a high-level policy offline that selects over a discrete set of pre-trained skills to perform effective control without additional online planning. Specifically, we use the discrete set of predictions from our multimodal forecasting model from Chapter 4 to represent a discrete set of observation-conditioned skills for driving. Then, we use our traffic forecasting model as a world model to explore a variety of counterfactual simulations offline in order to learn the high-level policy that selects over these skills. We find that for training this high-level policy, using stochastic and closed-loop simulations to evaluate different counterfactual behaviors for the ego-vehicle leads to the best performance compared to learning from other simulation methods like “log-replay.” Ultimately, we find that our approach to learning a hierarchical motion planning policy scales well with real-world driving data and achieves state-of-the-art results on the extensive nuPlan [40] benchmark for a solely learning-based policy when evaluated in closed-loop reactive simulation. Additionally, we find that our planning approach from Chapter 4 achieves results that are competitive on nuPlan [40] with other hybrid approaches that use learning to guide online planning.

Finally in Chapter 6, we conclude by discussing some high-level takeaways from our various projects and discuss some interesting directions for future work.

CHAPTER 2

Offline Learning with Conservative Counterfactuals

An appealing alternative to online data collection is to use learned models to collect additional hallucinated data in a model-based policy optimization [51] or Dyna [105] framework. This could allow the agent to explore different counterfactual alternatives beyond those previously experienced in the dataset. The main issue is that given limited data, these learned models can fail to generalize to different ego actions or fail to capture the full distribution of potential environment responses. Thus, these hallucinated trajectories can easily diverge from reality and cause the agent to learned a biased and suboptimal policy.

In this chapter, we present an algorithmic framework that combines ideas from behavior-regularization and uncertainty-aware model-based learning into an approach that allows us to effectively train a policy with both offline data and model-based hallucinated trajectories. Specifically, we first train a policy using behavior-regularized model-free RL. Then, we fine-tune our results with our algorithm Model-Based Behavior-Regularized Policy Optimization (MB2PO). We find that our approach is able to combine the upside of these approaches and achieve competitive or superior results on most of the Gym-MuJoCo [108] tasks in the D4RL [36] benchmark.

2.1 Related Work

While there exist many off-policy RL methods that can learn to solve a large variety of complex control tasks and can scale with large amounts of online data collection, these methods often perform quite poorly when run completely offline without any online data collection. Recently, there have been several methods that made progress in improving the capabilities of offline RL. For a general overview of the field of offline RL, we refer the reader to Levine et al. [64]. Here we will discuss some recent works that are particularly relevant to our approach.

2.1.1 Behavior-Regularized Model-Free RL

A variety of recent offline RL approaches have incorporated constraints or penalties on the learned policy's divergence from the empirical behavioral policy. However, most behavior-regularization or policy-constraint methods require the behavioral policy to be represented explicitly in order to estimate these divergences or to enforce their policy constraint [63]. In contrast, AWAC [74] or CRR [115] is able to incorporate a KL divergence constraint without explicitly representing the behavioral policy. They do this by reformulating the policy-constrained RL optimization equations into a form that resembles behavioral cloning re-weighted by the exponential of the advantage. Wang et al. [115] demonstrates that this method can effectively learn complex control tasks purely from offline data, and Nair et al. [74] demonstrate that performance can even be improved with further online data collection. In this work, we demonstrate that these properties make AWAC work exceptionally well when used for initialization as well as when used for fine-tuning with model-based policy optimization (MBPO) [51].

2.1.2 Uncertainty-Aware Model-Based RL

Model-based (MB) RL algorithms have several natural advantages for offline RL compared to model-free RL algorithms. First, MB RL algorithms rely on supervised learning, which provides more robust gradient signals compared to bootstrapped learning and policy gradients. Second, learning a dynamics model often provides strong task-independent supervision, which allows MB RL algorithms to learn from sub-optimal trajectories. These benefits make generalization easier, and can allow MB RL algorithms to surpass the performance of the demonstrated data. In fact, in many environments, MB RL methods have already been effective in learning with offline or randomly collected datasets. Recently, incorporating uncertainty estimation techniques from supervised learning in MB RL has demonstrated further improvement in both online [20] and offline deep RL. In particular, two recent works, Model-Based Offline Policy Optimization (MOPO) [124] and Model-Based Offline Reinforcement Learning (MoREL) [55], have demonstrated impressive results by incorporating uncertainty-aware MB RL with the Dyna [105] style algorithm MBPO [51]. Both methods use these models to create conservative MDPs that have a lower potential expected sum of rewards compared to the true MDP. By performing policy optimization in the conservative MDP through MBPO they are able to learn a conservative policy that can outperform the demonstrated trajectories. However, these methods can often fail to recover the expert policy even though it was demonstrated in the dataset. We believe that this is largely due to a lack of effective methods for estimating epistemic uncertainty for neural network regression.

2.2 Preliminaries

In RL, we treat the environment as a Markov decision process (MDP) represented as $(\mathcal{S}, \mathcal{A}, T, \mathcal{R}, \rho_0, \gamma)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ denotes the action space, $T(s'|s, a)$ represents the probabilistic transition dynamics, $\mathcal{R}$ is the reward function, ρ_0 is the initial state distribution, and $\gamma \in (0, 1)$ is the discount factor.

For MDPs, a trajectory is a sequence of states and actions $\tau = \{s_0, a_0, s_1, a_1, \cdots, s_T, a_T\}$. Each trajectory has corresponding rewards $r_\tau = \{r_0, \cdots, r_T\}$. The discounted return for a specific timestep is $R_t = \sum_{i=t}^{T} \gamma^{t-i} r_i$. The goal in RL is to find a policy that maximizes the expected discounted return $\mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$.

In offline RL, we are a given a fixed dataset $\mathcal{D}_\beta$ consisting of trajectories collected by some behavioral policy π_β in the environment. Without collecting additional data, we must learn a policy that will be effective immediately upon deployment.

2.3 Model-Based Behavior-Regularized Policy Optimization for Offline Fine-Tuning

For many offline datasets, it could be much harder to learn an effective model of the MDP than to learn a reasonable policy. This is especially the case when there is low variability or insufficient coverage of the state and action space in the collected dataset, or in environments with complex observations, like images, or long horizons. To overcome these issues, recent works [124; 55] have leveraged uncertainty estimation methods in order to construct conservative MDPs that use soft penalties or hard thresholds on model uncertainty to discourage deviating from the confident regions. However, these methods rely on the efficacy of ensemble-based neural network uncertainty estimation methods, which currently are not particularly effective at estimating epistemic uncertainty in regression settings. Therefore, we propose Model-Based Behavior-Regularized Policy Optimization (MB2PO). In MB2PO, we follow the offline uncertainty-aware MB framework of MOPO, but use the behavior-regularized model-free algorithm AWAC (also known as CRR-exp) instead of SAC [41] for policy optimization.

2.3.1 Conservative MBPO

In this work, we use MOPO [124] as a basis for our conservative MBPO, due to its simplicity and prior effective results on the D4RL benchmarks [36]. In MOPO, they construct a conservative MDP by augmenting the reward function as follows

$$\tilde{r}(s,a) = \hat{r}(s,a) - \lambda u(s,a)$$

where $\hat{r}$ is the learned estimate of the reward and u is the estimated uncertainty for the model transition. Note that this general formulation for a conservative MDP has also been explored in other prior work such as [37]. Still, we specifically follow MOPO in using the maximum standard deviation across an ensemble of probabilistic dynamics models as our measure of uncertainty.

While in theory, with well-calibrated uncertainty estimates and a proper tuning of λ, this should lead to only safe policy improvements over the behavioral policy, in practice it seems that MOPO is often unable to recover expert-level performance when it is provided in the offline dataset. This is unsurprising given that it is hard to generate well-calibrated epistemic uncertainty estimates in regression settings, and there will inevitably be model errors that will lead to overestimated Q-values.

To address these issues, we use policy-constrained model-free RL in MB2PO. In policy-constrained model-free RL, we attempt to optimize the following policy objective

$$\begin{aligned} \pi &=_{\pi} \mathbb{E}_{a\sim\pi(\cdot|s)}[Q^{\pi}(s,a)] \\ \text{s.t. } & D_{\text{KL}}(\pi(\cdot|s)\|\pi_{\beta}(\cdot|s)) \leq \epsilon \end{aligned} \tag{2.1}$$

If we estimate both π and π_β to be roughly Gaussians with similar variances, then the KL constraint becomes an ℓ_2 constraint on the policy mean. Because we expect our models to be locally accurate around the data, this constraint can help ensure that we stay in the effective region of the estimated MDP even if we have poorly calibrated uncertainty estimation. Additionally, Janner et al. [51] demonstrated that the difference between the true expected returns $J(\pi)$ and the expected returns $\hat{J}(\pi)$ of an MDP induced by an approximate model can be bounded by

$$J(\pi) \geq \hat{J}(\pi) - \left[\frac{2\gamma r_{\max}(\epsilon_m + 2\epsilon_\pi)}{(1-\gamma)^2} + \frac{4 r_{\max}\epsilon_\pi}{1-\gamma}\right]$$

where $r_{\max}$ is the maximum reward, γ is the discount factor, ϵ_m is a bound on the total variation distance (TVD) between the learned model and the true model, and ϵ_π is a bound on the TVD between π and π_β on the demonstrated states. By Pinker's inequality, bounding the KL divergence also bounds the TVD. Therefore, by leveraging policy constraints in the policy optimization in MBPO, we can reduce the gap in expected returns and improve the algorithm's robustness to model errors.

2.3.2 Behavior-Regularized Model-Free RL with AWAC

For performing behavior-regularized policy optimization, we use AWAC [74], also known as CRR-exp [115] due to its impressive results in offline RL and its ability to be fine-tuned with additional online data.

By enforcing the KKT conditions [79; 81; 39], we can derive an analytic solution to Equation 2.1, where the Lagrangian is

$$\mathcal{L}(\pi, \alpha) = \mathbb{E}_{a\sim\pi(\cdot|s)}\left[Q^\pi(s,a)\right] + \alpha(\epsilon - D_{\mathrm{KL}}(\pi(\cdot|s)\|\pi_\beta(\cdot|s)))$$

We can substitute $A^\pi(s,a)$ for $Q^\pi(s,a)$ because it does not affect the optimum and get the closed-form solution

$$\pi^*(a|s) = \frac{1}{Z(s)}\pi_\beta(a|s)\exp\left(\frac{A^\pi(s,a)}{\alpha}\right)$$

where $Z(s)$ is the normalizing partition function. In order to project this solution into our policy space, we update our parameters by minimizing $D_{\mathrm{KL}}(\pi^*\|\pi_\theta)$. This leads to the following iterative update

$$\theta_{k+1} =_\theta \mathbb{E}_{s,a\sim D}\left[-\log\pi_\theta(a|s)\frac{1}{Z(s)}\exp\left(\frac{A^{\pi_k}(s,a)}{\alpha}\right)\right] \tag{2.2}$$

One of the major benefits of using AWAC is that we can leverage behavior regularization in a principled manner without needing to explicitly represent the behavioral policy. This is particularly important in 3 major cases: 1. when there are not enough data to learn the behavioral policy; 2. when the data were collected by a variety of different policies or sources; 3. when the data were collected by a policy outside of your policy class, such as a human expert or a controller that leverages hidden state information.

Additionally, we can view AWAC as a reweighted behavioral cloning algorithm. Unlike SAC [41] and DDPG [65], it does not rely on the reparametrization trick or gradients of your learned Q-function to perform policy updates. This allows us to use a wider ranger of policy classes, which in this work we take advantage of by using a tanh-squashed GMM with 5 components. We suspect that there are also some additional benefits to not depending on the gradients of the learned Q-function, which might be particularly bad in offline settings, but leave further investigation to future work.

An important thing to note with AWAC is that we can influence the implicit behavioral penalty by controlling the source of the data we train with. This holds, for example, if we perform a series of policy updates only using data collected by the previous policy iterate. Then, we are implicitly performing a trust-region policy update like TRPO [96] and PPO [97] of the form

$$\begin{gathered}\pi_{k+1} =_{\pi} \mathbb{E}_{a\sim\pi(\cdot|s)}[Q^{\pi_k}(s,a)] \\ \text{s.t. } D_{\mathrm{KL}}(\pi(\cdot|s)\|\pi_k(\cdot|s)) \le \epsilon\end{gathered}$$

In fact, if we train on data collected by the last n policy iterates, then we are approximately constraining our policy to a weighted sum of the previous n policies $\pi_k^{(n)} = \frac{1}{n}\sum_{i=0}^{n-1}\pi_{k-i}$ and damping our learning process in the policy space.

In our work, we train with a $\omega \in [0,1]$ portion of the data from offline data collected by π_β and a $(1-\omega)$ portion of the data collected online from the last n policy iterates in the conservative MDP defined by our learned models. Therefore, we are approximately optimizing the following objective

$$\mathbb{E}_{a\sim\pi(\cdot|s)}\left[\hat{Q}^{\pi}(s,a)\right] - \alpha\left(\omega D_{\mathrm{KL}}(\pi(\cdot|s)\|\pi_\beta(\cdot|s)) + (1-\omega)D_{\mathrm{KL}}(\pi(\cdot|s)\|\pi_k^{(n)})\right)$$

Therefore, by using AWAC as the policy optimization algorithm in MB2PO, we can easily perform behavior-regularized policy optimization with soft damped trust region updates in the conservative MDP to reduce the effects of model errors and poor uncertainty estimation.

2.3.3 Model-Based Behavior-Regularized Policy Optimization

We first initialize our policy by training with AWAC solely on the offline data. Next for fine-tuning with MB2PO, we train an ensemble of probabilistic dynamics models represented by neural networks that output a diagonal Gaussian distribution over the next state and reward:

$$\{\hat{T}_\theta^i(s_{t+1}, r|s_t, a_t) = \mathcal{N}(\mu_\theta^i(s_t,a_t), \Sigma_\theta^i(s_t,a_t))\}_{i=1}^N$$

We construct a conservative MDP that at every time step uses a randomly drawn dynamics model from $\{\hat{T}^i_\theta\}_{i=1}^M$ to determine the next state transition. Additionally, we incorporate a penalty on the largest predicted standard deviation among the dynamics models as a practical means of penalizing both epistemic and aleatoric uncertainty.

Algorithm 1: MB2PO

```
Train $\pi_\theta$, $Q_\phi$ with AWAC with samples from $\mathcal{D}_\beta$
Train an ensemble of N probabilistic dynamics
  $\{\hat{T}^i_\theta(s_{t+1}, r|s_t, a_t) = \mathcal{N}(\mu^i_\theta(s_t, a_t), \Sigma^i_\theta(s_t, a_t))\}_{i=1}^N$ on the data in $\mathcal{D}_\beta$
for epoch k= 1, 2, ... do
    Initialize empty replay buffer $\mathcal{D}_k$
    for 1, 2, ..., batchsize do
        Sample state $s_1$ from $\mathcal{D}_\beta$
        for j = 1, 2, ..., h do
            $a_j \sim \pi(s_j)$
            Uniformly sample $\hat{T}$ from $\{\hat{T}^i\}_{i=1}^N$
            $s_{j+1}, r_j \sim \hat{T}(s_j, a_j)$
            $\tilde{r}_j = r_j - \lambda \max_{i=1}^N \|\Sigma^i(s_j, a_j)\|_F$
            Add sample $(s_j, a_j, \tilde{r}_j, s_{j=1})$ to $\mathcal{D}_k$
        end
    end
    Draw $\omega$ portion of the samples from $D_\beta$ and the rest uniformly from
      $\{\mathcal{D}_{k-i}\}_{i=0}^{99}$ to train $\pi_\theta$ and $Q_\phi$ with AWAC
end
```

Then, we alternate between collecting data with our current policy in the conservative MDP and updating our policy and Q-network. When collecting data in the conservative MDP, we collect h-length truncated trajectories starting from states in the original offline dataset. By collecting data this way, we are able to collect a variety of imagined data without relying on long model rollouts, which would inevitably lead to compounding errors. When performing training updates, we sample $\omega \in [0, 1]$ of the data from the original dataset and the remaining $1 - \omega$ uniformly from the last 100 policy iterates. Our full algorithm is outlined in Algorithm 1.

2.4 Experiments

In our experiments, we want to demonstrate that we can further improve performance by fine-tuning with MB2PO. We evaluate this by fine-tuning the policy and Q-function, after running AWAC for 500000 gradient steps, with MB2PO. In particular, we compare our re-

Task and Dataset	AWAC	**AWAC+ MB2PO (Ours)**	MOPO	BEAR	BRAC	CQL
halfcheetah-random	18.8 ± 1.5	25.5 ± 1.1	**35.4** ± 2.5	25.5	28.1	**35.4**
hopper-random	11.2 ± 0.1	11.4 ± 0.1	**11.7** ± 0.4	9.5	12.0	10.8
walker2d-random	1.4 ± 3.0	0.2 ± 2.3	**13.6** ± 2.6	6.7	0.5	7.0
halfcheetah-medium	40.9 ± 0.3	40.7 ± 0.2	42.3 ± 1.6	38.6	**45.5**	44.4
hopper-medium	35.0 ± 3.9	55.7 ± 14.6	28.0 ± 12.4	47.6	32.3	**58.0**
walker2d-medium	74.3 ± 1.6	80.4 ± 0.9	17.8 ± 12.7	33.2	**81.3**	79.2
halfcheetah-expert	106.7 ± 0.6	105.1 ± 1.3		**108.2**	-1.1	104.8
hopper-expert	108.1 ± 3.8	105.5 ± 10.5		**110.3**	3.7	109.9
walker2d-expert	100.5 ± 12.3	107.4 ± 1.1		106.1	0.0	**153.9**
halfcheetah-med-expert	104.7 ± 1.6	**104.8** ± 1.1	63.3 ± 38.0	51.7	45.3	62.4
hopper-med-expert	75.1 ± 15.9	79.1 ± 13.5	23.7 ± 6.0	4.0	0.8	**111.0**
walker2d-med-expert	81.8 ± 18.9	86.2 ± 35.5	44.6 ± 12.9	26.0	66.6	**98.7**
halfcheetah-mixed	42.3 ± 0.3	**55.6** ± 0.6	53.1 ± 2.0	36.2	45.9	46.2
hopper-mixed	30.1 ± 0.9	**72.6** ± 25.5	67.5 ± 24.7	25.3	0.8	48.6
walker2d-mixed	16.9 ± 1.7	**61.9** ± 8.2	39.0 ± 9.6	10.8	0.9	26.7

Table 2.1: Here we compare AWAC (averaged over 4 seeds) and AWAC + MB2PO (Ours) (averaged over 4 seeds) to recent offline model-free and model-based RL algorithms. We report the normalized score where 100 is the performance of a fully trained SAC policy and 0 is the performance of a uniform random policy. For the other methods, we report the results from their own papers or the original D4RL paper. "-expert" results for MOPO were not included in the original paper and thus are omitted here. We include the standard deviation for our results and for previous results if reported. We bold the highest mean.

sults to other model-free offline RL algorithms: BRAC-v [120], BEAR [61], and CQL [62] on the Gym-MuJoCo tasks [108] in the D4RL benchmark. In addition, we also compare these results to the MB offline RL algorithm MOPO.

Results in Table 2.1 demonstrate that AWAC on its own can get reasonable results on all the datasets and can approach state-of-the-art results on "-expert" and "-med-expert" datasets. Unlike the other behavior-regularized model-free methods, AWAC and CQL are able to get near or fully recover expert-level performance when trained on the "med-expert-" datasets. This indicates that AWAC and CQL are more robust, as there is less of a drop in performance compared to other methods when incorporating additional sub-optimal trajectories.

Next, we fine-tune the trained AWAC policies with MB2PO. For each task and dataset, we pretrain an ensemble of 5 probabilistic dynamics models for 100000 gradient steps on the behavioral dataset. We then perform MB2PO for 500 iterations. Each iteration consists of collecting 1000000 steps from h-length truncated trajectories in the conservative MDP, which should run in a few seconds on modern GPU hardware, followed by 1000 AWAC gradient steps.

Results in Table 2.1 demonstrate that our method is effective in improving the performance over AWAC in 11 of the 15 tasks. In particular, we find that MB2PO significantly improves the performance on all of the "-mixed" datasets. These strong results in the "-mixed" datasets demonstrate that our model-based fine-tuning method can be especially beneficial when there is sufficient variation in the behavioral dataset. Additionally, the noticeable improvement in some of the "-medium" and "-med-expert" datasets demonstrates that our fine-tuning can be effective even when the data were collected by one or two policies. In the 4 cases where MB2PO fine-tuning degrades performance, it is always negligible and never over 3 points.

Our method is quite competitive with CQL, as we beat it for 7 of the 15 tasks, and generally our results are quite comparable. One important note is that our method significantly outperforms CQL in all of the "-mixed" datasets. These results indicate the potential benefits of leveraging learned models when there is variety in the offline dataset.

Finally, our method outperforms MOPO, the most direct comparison, in 8 out of the 12 comparable tasks. These results demonstrate the benefits of combining behavior-regularized model-free RL with uncertainty-aware MB RL, as we are able to get the best of both worlds. We are able to recover high-level performance when available like the "-expert" and "-med-expert" datasets, and we can still generalize and learn to outperform the best observed trajectory, as demonstrated in the "-medium" and "-mixed" datasets.

2.5 Conclusion

In this chapter, we proposed an algorithmic framework that leverages the benefits of both behavior-regularized model-free methods and uncertainty-aware model-based methods. We do this by first training an initial policy with the offline model-free AWAC algorithm. Then, we fine-tune with our MB2PO algorithm. We perform this by learning uncertainty-aware models that are used to create a conservative MDP. Then, we continue to use AWAC to further update our policy and Q-function in this conservative MDP. By using AWAC, we are able to perform policy optimization while implicitly constraining the learned policy's KL divergence to the behavioral policy. We demonstrate that this two-stage process allows us to get the best of both worlds between behavior-regularized model-free methods and uncertainty-aware model-based methods. Specifically, the initial AWAC training allows us to often recover the best-performing behavior in the dataset, and MB2PO fine-tuning can allow us to generalize and outperform the demonstrated behavior.

CHAPTER 3

Addressing Optimism Bias in Sequence Modeling for Reinforcement Learning in Stochastic Environments

Our work in the prior chapter and most related works in offline RL have focused on the mainly deterministic D4RL [36] benchmarks and a variety of weakly stochastic Atari [5] benchmarks. Therefore, there has been limited focus on the difficulties of deploying such methods in largely stochastic domains. In this work, we instead focus specifically on stochastic safety-critical domains like autonomous driving. In autonomous driving, understanding the stochastic and multimodal nature of traffic is critical to safe and robust performance. For example, the same turning sequence for the ego-vehicle could lead to a successful traversal of an intersection or a crash depending on the unknown intentions of the other agents. In these stochastic settings, it is important to disentangle the effects of the policy and the world dynamics when evaluating different potential outcomes. In this chapter, we will present our work that does explicit disentangling of the policy and world models, which allows us at test time to search for policies that are robust to multiple possible futures in the environment.

3.1 Introduction

Recent works have achieved impressive results in a variety of sequence modeling problems from NLP [10; 86; 85] to trajectory prediction [77; 68; 84; 125] by leveraging powerful Transformer [112] models. Inspired by these numerous successes, several recent works [16; 52] have explored ways of reformulating sequential decision-making problems in the offline RL framework as a single sequence modeling problem. In particular, these approaches jointly model the states, actions, and rewards as a single data stream with a high-capacity Transformer. These Transformer-based methods are able to outperform the behavioral policy at test time by either conditioning on desired outcomes when picking actions or leveraging the model to search for high-reward trajectories. The main benefit of this paradigm is that it avoids many of the complexities involved in modern model-free and model-based offline RL algorithms.

There are 2 major issues with treating RL as a single sequence modeling problem like the words or characters in a NLP problem. **(1)** States and actions are fundamentally different concepts. The agent always has complete control over its action sequences, but often has only limited influence on the resulting state transitions. In adversarial or stochastic environments, the same action in the same state can lead to potentially different outcomes, which affects the likelihood or feasibility of achieving a desired result. This leads to the more practical issue **(2)**, which is that we often need to perform different optimizations over the policy actions and the potential state transitions. Generally, we want to find the action that maximizes reward, but either in expectation over possible future states or with respect to the worst case scenario. Thus, in safety-critical domains or adversarial games we often want to perform a maximum over potential actions and a minimum over possible futures in the environment. Thus in these types of environments, special considerations are needed to ensure effective planning during deployment. Ozair et al. [78] demonstrates similar issues when deploying MuZero [95] with different MCTS [23] frameworks in chess. They find that planning with a single-player variant of MuZero that treats the other player as an unknown part of the environment results in a catastrophic drop in performance relative to the traditional two-player adversarial framework.

In stochastic or adversarial settings, the deployment strategies described in recent sequence modeling approaches for offline RL will lead to overly optimistic behavior because they do not properly disentangle the effects of the policy and world dynamics on the outcome. Specifically, they are biased to believe that the environment will cooperate with them because those sequences are most likely to lead to desired high returns. In order to address this optimism bias in prior approaches, we develop a method called SeParated Latent Trajectory Transformer (SPLT Transformer), which learns separate generative Transformer models for the policy and dynamics. Because we focus on the autonomous driving domain, we represent each of these models as a discrete latent variable Variational Auto-Encoder (VAE) [57] as inspired by prior work [106]. By training two separate discrete latent variable VAE models, we can efficiently search over different possible ego behaviors and their interactions with different possible environment responses. We demonstrate how this planning approach allows us during test-time to search for policies that are robust to many possible futures in the environment.

3.2 Related Work

Many recent works have explored leveraging high-capacity sequence models in sequential decision making problems as well as in autonomous driving. Most of the works in the former area have focused on deterministic environments and thus struggle in stochastic and multimodal problems like autonomous driving. Many of the works applying sequence models to autonomous driving have focused specifically on joint trajectory forecasting for all the vehicular and pedestrian agents in the scene. These methods have been effective in capturing the multimodal stochastic nature of autonomous driving, but often do not consider how these approaches should be leveraged to generate a robust driving policy during deployment. In this work, we develop a method that learns separate sequential policy and dynamics models entirely from offline data. Then, we demonstrate how these models can be leveraged at test-time to perform robust planning solely from quantities inferable from an ego-centric perception system.

3.2.1 Sequence-Modeling for Offline RL

Two major works that have explored reformulating the offline RL problem as a single sequence modelling problem are Decision Transformer [16] and Trajectory Transformer [52]. In stochastic environments like autonomous driving, both approaches can act optimistically because they fail to account for the uncontrollable factors in the environment that can affect the ability to achieve a specific return.

Decision Transformer is a return-conditioned model-free method that learns a Transformer-based policy that takes in the historical states and actions and a target return, and outputs the action that is most likely to lead to a trajectory sequence that achieves the target return. In stochastic environments, the difficulties come from picking a suitable target return without strong prior knowledge of the testing domain. This is especially problematic because the distribution of possible returns is heavily dependent on the stochastic transitions of the environment. Thus, setting a large target return that is not always feasible can lead to overly aggressive and optimistic behavior, while setting any lower target return could lead to sub-optimal behavior in situations where the environment does unroll favorably.

Trajectory Transformer is a model-based method that trains a Transformer-based trajectory model that can hallucinate potential trajectories in the environment. In stochastic environments, the main concern is how you use this model to properly search for a suitable next action or trajectory-sequence given the uncontrollable stochastic transitions. Naively deploying NLP-style beam search as proposed in the original Trajectory Transformer paper [52] without accounting for these uncontrollable stochastic transitions will be biased to explore and pick trajectories where the environment just so happens to unroll favorably. This will similarly lead to optimistic behavior, which can be dangerous in safety-critical environments. To address this issue, a different searching procedure needs to be used that can reason about the states and actions separately. One possible avenue for doing this step-wise would be to explore the different variants of MCTS described in Ozair et al. [78], but that would lead to an approach that scales exponentially with the search horizon, while our approach only scales linearly with the search horizon.

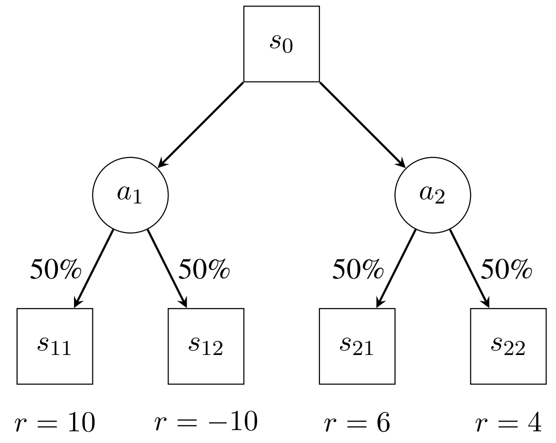


Figure 3.1: Simple stochastic MDP where being optimistic leads to suboptimal behavior.

In order to illuminate the optimism bias in Decision Transformer and Trajectory Transformer, consider the simple discrete and stochastic MDP in Figure 3.1 where there are only 5 states $\{s_0, s_{11}, s_{12}, s_{21}, s_{22}\} \in \mathcal{S}$ and 2 actions $\{a_1, a_2\} \in \mathcal{A}$. The agent always starts in s_0. Then, if the agent takes a_1 it will stochastically transition from $s_0 \to s_{11}$ or $s_0 \to s_{12}$ with equal probability. Similarly, if the agent takes a_2 it will stochastically transition from $s_0 \to s_{21}$ or $s_0 \to s_{22}$ with equal probability. The reward at each state is: 0 for s_0, 10 for s_{11}, -10 for s_{12}, 6 for s_{21}, and 4 for s_{22}. The trajectory terminates upon reaching any state besides s_0. In this MDP, the expected return for taking a_1 is 0 and for taking a_2 is 5. Let's assume that our dataset consists of samples from a uniform random policy. Now, the optimism bias in Trajectory Transformer comes from the modified beam search they use at test-time. They jointly unroll the next action and resulting state, reward, and return and then filter based on which trajectories have the highest estimated return. In this setting, the trajectory model should properly predict all possible transitions. Then, their beam search will chose to take action a_1 because it will lead to the highest return of 10 when it predicts the transition $s_0 \rightarrow s_{11}$. However, picking a_1 is obviously suboptimal in expectation, even if a_1 could lead to the best possible return. The optimism bias for Decision Transformer comes from picking a fixed target return and their default heuristic of setting it to be the highest in the dataset. If we condition on getting a return of 10 (the highest possible), then it will clearly pick action a_1 as that is the only action where the probability of achieving the desired outcome is nonzero. Note that this is an issue for any return-conditioned method, not just Decision Transformer. The inclination for both methods to select a_1 because there is a possibility it has the highest return, even though a_2 is a better action in expectation and in the worst-case scenario is the specific optimism bias we allude to in this proposal.

We validate the optimism bias in these approaches in a toy multimodal autonomous driving task, and further demonstrate how addressing these issues leads to our method achieving superior performance.

3.2.2 Conservatism and Risk-Sensitive RL

Our work is also related to the fields of risk-sensitive RL and conservatism for offline RL. Most prior works in risk-sensitive RL focus on learning a policy that is optimized not for expected return, but rather some risk-sensitive profile over the distribution of returns [72; 100]. Many approaches accomplish this by learning a distributional Q-function that estimates the full distribution of potential returns [26] rather than just the expected return. Instead of learning a distributional or risk-sensitive Q-function, we use a learned Transformer world model to hallucinate possible future trajectories and pick a robust behavior that does well in the worst-case predicted future.

Prior works in offline RL have explored different methods of incorporating conservatism in both model-based [124; 55] and model-free [62; 58] RL methods. However, offline RL approaches generally leverage conservatism specifically to discourage the agent from visiting state-actions that are outside the training distribution where the learned models could fail to generalize [64]. In contrast, in this work we explore using a conservative approach to address the difficulties of planning in stochastic environments in safety-critical domains, like autonomous driving.

3.2.3 Trajectory Prediction in Autonomous Driving

Many recent works in trajectory prediction leverage attention-based models [106; 19] and in particular Transformer neural networks [77; 68; 84; 125] to make accurate long-term predictions in complex traffic situations. These methods can learn to attend specifically to the relevant factors in both the target and surrounding vehicles' recent trajectories in order to make predictions that are consistent with the surrounding traffic. Additionally, many trajectory prediction approaches have incorporated VAEs [106; 84; 125] to facilitate covering the different possible modes of future traffic trajectories. However, these approaches have mostly been evaluated on different variations of prediction error on a held out test-set with little focus on the effectiveness of leveraging these models for long-term online planning, like for the CARLA [31] NoCrash [22] or Leaderboard[1] benchmarks. In this work, we also leverage Transformers and VAEs in order to make multimodal trajectory predictions. However, we focus specially on learning models that can be effectively used for robust search during deployment on an ego vehicle.

Still, there are some works that have leveraged multi-agent trajectory prediction for online planning in simulation. Rhinehart et al. [92] explores similar problems with planning interactive behaviors under uncertainty, but in different multi-agent planning frameworks for autonomous driving. We focus, on the other hand, on addressing the optimism bias that arises in prior single-agent Transformer-based offline RL approaches by training separate world and policy latent models with discrete time-consistent latent variables, so that we can do robust planning efficiently. Cui et al. [24] also evaluates using closed-loop simulation, but only for a limited horizon of 18 seconds. Additionally, they do not incorporate a learned ego-vehicle policy in planning like our work, and instead do MPC with a learned cost function evaluated over different predicted potential traffic scene futures.

3.2.4 Learning Behavior for Self-Driving

Recently, there have been several works that have focused on learning behavioral policies for autonomous driving from offline datasets. Most of this work revolves around performing imitation learning on the behavioral policy of a privileged autopilot agent in simulation [14; 83; 15]. In our work, we also use a privileged autopilot to collect our offline dataset in simulation. Instead of just learning the behavioral policy, we use these data to learn both a multimodal policy, and a multimodal world model that allows us to do efficient and robust search at test-time. Later, we demonstrate how this search leads our method to outperform pure imitation learning.

[1]https://leaderboard.carla.org/

3.3 Preliminaries

3.3.1 Transformers

Transformers [112] are a neural network architecture that use several stacks of self-attention blocks to process an arbitrary collection of inputs. By leveraging positional encodings and a causal-attention map, Transformers can be used as sequence models for autoregressive generation. In this work, we mostly use a similar GPT-based [85] architecture as DT [16]. In particular, we use the same linear layer + layer normalization to project the raw inputs into the embedding dimension. We use 4 self-attention blocks with 8 self-attention heads. While our high-level details are more inspired by DT, our code and GPT implementation are based on the publicly available TT [52] codebase[2].

3.3.2 Variational Auto-Encoder

Our method uses conditional variational autoencoders (CVAE) [102] as generative policy and world models that we can use to generate realistic and multimodal candidate trajectories for test-time search. Thus, in this section we give a general overview of variational autoencoders (VAE) [57]. The goal of a VAE model is to generate samples $\hat{d}$ that are within the distribution of a training set $\mathcal{D} = \{d_i\}_{i=1}^{N}$. Thus the model should be trained to maximize the likelihood of all training points d_i. VAEs accomplish this by sampling a latent variable z from some prior $p(z)$ and training a decoder $p_\theta(d|z)$ to convert the latent variable into a sample. However, optimizing this model is often intractable and thus Kingma and Welling [57] introduces an encoder $q_\phi(z|d)$ that can be used to instead optimize the evidence lower bound (ELBO) on the log-likelihood of the data

$$\log p(d) \geq E_{z \sim q_\phi(z|d)}[\log p_\theta(d|z)] - \mathcal{D}_{KL}(q_\phi(z|d)||p(z))$$

The expectation term represents the reconstructions loss, and the KL Divergence term acts as a regularizer that keeps the encoder output close to the prior. Thus with a trained VAE model, we can sample a z from our prior $p(z)$ and pass it to our decoder $p_\theta(d|z)$ to generate a sample from the desired distribution.

β-VAE When training VAEs with high-capacity models like Transformers or CNNs [60], the decoders often learn to ignore the high-entropy stochastic latent variables. Therefore, Higgins et al. [45] proposed a modification to the standard ELBO loss and introduced

[2]https://github.com/JannerM/trajectory-transformer

a coefficient β to the KL Divergence term. Lowering this hyperparameter β below 1 reduces the regularizing effect on the latent variables and thus makes it easier for the models to incorporate the latent variables into their predictions. We find $\beta \in [0.0001, 0.01]$ to be effective for our experiments.

3.4 Separated Latent Trajectory Transformer

In this section, we describe how we train two separate Transformer-based discrete latent variable VAEs to represent our policy and world models, and how we leverage these models for robust planning at test-time. Figure 3.2 contains an overview of the entire SPLT model architecture.

3.4.1 Discrete Latent Variable VAE

In order for our models to be useful for search, they need to **(1)** be able to produce a good range of candidate behaviors for any given situation for the ego vehicle and **(2)** cover a majority of the different modes of potential responses from the agents in the environment. Towards this end we train separate Transformer-based VAEs for both the policy and world models. We make the specific design choice for the stochastic latent variables for both models to be discrete and consistent over the entire planning horizon. This allows us to tractably enumerate all possible candidate trajectories without exponential branching, which enables efficient search at test-time. The intuition is that the policy latent variables should correspond to different high-level intentions or policies for the ego-vehicle, like whether to tail another vehicle aggressively or keep your distance. Similarly, the world latent variables should correspond to different possible intentions of the observable vehicles, like whether a vehicle in the opposing lane will go straight or turn through the intersection. Additionally, the world latent variables should capture other events like lights changing or cars suddenly appearing in the sensing range after rounding a corner.

3.4.2 Encoders

Both the world encoder q_{ϕ_w} and policy encoder q_{ϕ_π} use the same architecture and receive the same trajectory representation. Specifically, they both take in K-length trajectory sequences of the form $\tau_t^K = \{s_t, a_t, s_{t+1}, a_{t+1}, \cdots, s_{t+K}, a_{t+K}\}$ and output a n_w or n_π

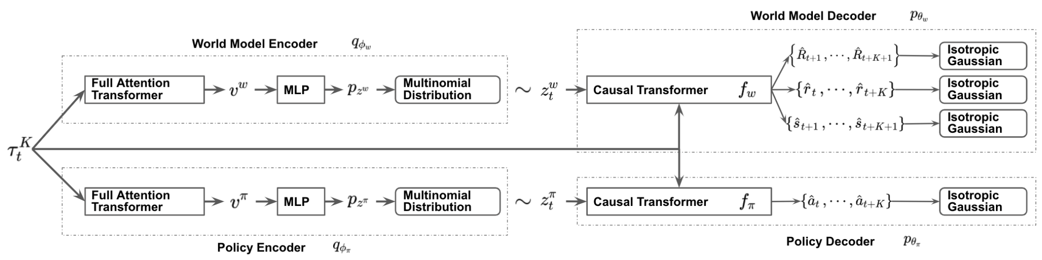

World Model Encoder q_{ϕ_w}
Full Attention Transformer
v^w
MLP
p_{z^w}
Multinomial Distribution
$\sim z_t^w$
World Model Decoder p_{θ_w}
Causal Transformer f_w
$\{\hat{R}_{t+1}, \cdots, \hat{R}_{t+K+1}\}$
$\{\hat{r}_t, \cdots, \hat{r}_{t+K}\}$
$\{\hat{s}_{t+1}, \cdots, \hat{s}_{t+K+1}\}$
Isotropic Gaussian
Isotropic Gaussian
Isotropic Gaussian
τ_t^K
Full Attention Transformer
v^π
MLP
p_{z^π}
Multinomial Distribution
$\sim z_t^\pi$
Causal Transformer f_π
$\{\hat{a}_t, \cdots, \hat{a}_{t+K}\}$
Isotropic Gaussian
Policy Encoder q_{ϕ_π}
Policy Decoder p_{θ_π}

Figure 3.2: Overview of the SPLT Transformer architecture for generating a reconstruction prediction. The World Model (top) attempts to reconstruct the discounted returns, rewards, and states while the Policy Model (bottom) attempts to reconstruct the action sequence.

dimensional discrete latent variable with each dimension having c possible values

$$\begin{aligned} z^w &\sim q_{\phi_w}(\cdot|\tau_t^K), & z^w &\in \{1, \cdots, c\}^{n_w} \\ z^\pi &\sim q_{\phi_\pi}(\cdot|\tau_t^K), & z^\pi &\in \{1, \cdots, c\}^{n_\pi} \end{aligned}$$

The core modules for both encoders are Transformers based on the GPT architecture similar to TT and DT, except we do not perform any masking of the attention. Thus, all elements in the Transformer can fully attend to every other component in the sequence. We take the mean of the Transformer outputs for all the elements in order to coalesce the entire trajectory into a single vector representation v^w and v^π. Finally, we pass each of these outputs into a small MLP that outputs the n_w and n_π independent categorical distributions for z^w and z^π respectively. Thus, the conditional distributions for z^w and z^π are represented as n_w and n_π independent multinomial distributions respectively. We leverage the straight-through gradient estimator [6] as described in Hafner et al. [42] in order to make the sampling procedure fully differentiable for training.

3.4.3 Policy Decoder

The policy decoder uses a similar input trajectory representation
$\tau_t'^k = \{s_t, a_t, s_{t+1}, a_{t+1}, \cdots, s_{t+k}\}$ and also takes in the latent variable z^π. The goal of the policy decoder is to estimate $p_{\theta_\pi}(a_{t+k}|\tau_t'^k; z^\pi)\ \forall k \in [1, K]$, so that we can predict the most likely next action in the trajectory. We represent this decoder with a causal Transformer model that is very similar to the ones used in Decision Transformer. Beyond excluding the returns-to-go as inputs, the main difference in our model is that we need to incorporate

the latent variable z^π. In this work, we incorporate z^π by first converting it into a single embedding vector, similar to the positional encodings used in other Transformer works, and add it to all the state and action embeddings. We make this design choice instead of inputting the latent variable as another element in the sequence because our method makes it harder for the decoder Transformer to learn to ignore the latent variable, which is a common issue when using high-capacity models in VAEs.

For simplicity, we represent our output distribution as a unit-variance isotropic Gaussian with the mean outputted by our deterministic decoder f_π

$$p_{\theta_\pi}(a_{t+k}|\tau_t'^k; z^\pi) := \mathcal{N}(f_\pi(\tau_t'^k, z^\pi), I)$$

3.4.4 World Model Decoder

The world model decoder is very similar to the policy decoder, except that its goal is to estimate $p_{\theta_w}(s_{t+k+1}|\tau_t^k; z^w), p_{\theta_w}(r_{t+k}|\tau_t^k; z^w), p_{\theta_w}(R_{t+k+1}|\tau_t^k; z^w)\ \forall k \in [1, K]$ so that we can predict the most likely next state, reward, and discounted return in the trajectory. The world model decoder is similarly represented with a causal Transformer and incorporates its latent variable z^w in the same manner as the policy decoder. The major difference is that the world model decoder has 3 separate heads to output the 3 different required quantities. The output distributions are similarly represented as unit-variance isotropic Gaussians with the means outputted by the different heads of the deterministic decoder f_w

$$\begin{aligned}
p_{\theta_w}(s_{t+k+1}|\tau_t^k; z^w) &:= \mathcal{N}(f_w^s(\tau_t^k, z^w), I) \\
p_{\theta_w}(r_{t+k}|\tau_t^k; z^w) &:= \mathcal{N}(f_w^r(\tau_t^k, z^w), I) \\
p_{\theta_w}(R_{t+k+1}|\tau_t^k; z^w) &:= \mathcal{N}(f_w^R(\tau_t^k, z^w), I)
\end{aligned}$$

3.4.5 Variational Lower Bound

In order to train our CVAEs, we wish to minimize the standard evidence lower bound (ELBO) on the log-likelihood of the behavioral data for the policy model

$$\mathbb{E}_{z^\pi \sim q_{\phi_\pi}}\left[\sum_{k=1}^{K} \log p_{\theta_\pi}(a_{t+k}|\tau_t'^k; z^\pi)\right] - \mathcal{D}_{KL}(q_{\phi_\pi}(z^\pi|\tau_t^K)||p(z^\pi)) \qquad (3.1)$$

and the world model

$$\mathbb{E}_{z^w \sim q_{\phi_w}}\left[\sum_{k=1}^{K} \log p_{\theta_w}(s_{t+k+1}, r_{t+k}, R_{t+k+1} | \tau_t^k; z^w)\right] - \mathcal{D}_{KL}(q_{\phi_w}(z^w | \tau_t^K) || p(z^w)) \tag{3.2}$$

The main difference between our CVAE formulation and the standard formulation is that we use independent discrete uniform distributions as our prior

$$\begin{aligned} p(z^\pi)_i &:= \mathcal{U}\{1, c\} & \forall i \in [1, n_\pi] \\ p(z^w)_i &:= \mathcal{U}\{1, c\} & \forall i \in [1, n_w] \end{aligned}$$

because our latent variables are discrete and multidimensional. This prior makes the KL-Divergence terms in equations (3.1) and (3.2) equivalent to independently maximizing the entropy of each dimension of z^π and z^w respectively. This regularizes the decoders and encourages the VAEs to leverage all available combinations of discrete latent variables.

Because we use the straight-through gradient estimator, we can differentiate through the multinomial sampling of the latent variables. Thus for both CVAEs, we can jointly train the encoders and decoders end-to-end by directly optimizing objective (3.1) for the policy models and objective (3.2) for the world models.

3.4.6 Training

During training, we sample batches of K-length trajectories from our offline dataset. The states and actions of these trajectories are passed into the world and policy encoders in order to generate the corresponding z^w and z^πs. Each z^π and its corresponding trajectory are passed into the policy decoder, where it predicts all actions in the trajectory through the standard teacher-forcing procedure [118]. Each z^w and its corresponding trajectory are passed into the world decoder, where it predicts all the next states, rewards, and discounted returns in the trajectory also through the teacher-forcing procedure. The policy decoder and encoder parameters are updated to minimize equation (3.1), and the world decoder and encoder parameters are updated to minimize equation (3.2). We train all our models with the Adam optimizer [56] with a learning rate of $1\text{e}{-4}$ and weight decay 0.1. Additionally, we normalize all raw values by subtracting the mean and dividing by the standard deviation of the dataset.

3.4.7 Planning

In this section, we describe how we can use our trained conditional policy and world model decoders in order to perform efficient and robust search at test-time.

First, we describe how we can generate a single candidate trajectory given a specific z^π and z^w. Assume that we are currently at a state s_t and we have stored a history of the last k steps of the trajectory ${\tau'}_{t-k}^{k} = \{s_{t-k}, a_{t-k}, s_{t-k+1}, a_{t-k+1}, \cdots, s_t\}$. Our goal is to predict a possible continuation of that trajectory over the planning horizon h, which corresponds to $\hat{\tau}_{t-k}^{k+h} = \{s_{t-k}, a_{t-k}, \cdots, s_t, \hat{a}_t, \hat{s}_{t+1}, \cdots, \hat{s}_{t+h}, \hat{a}_{t+h}\}$. Additionally, we want to estimate the future discounted returns for our candidate trajectory $\hat{R}(\hat{\tau}_t^h) = \sum_{i=0}^{h} \left[\gamma^i \hat{r}_{t+i}\right] + \gamma^{h+1} \hat{R}_{t+h+1}$.

In order to predict these quantities, we alternatively make autoregressive predictions from the policy and world models. Specifically, we alternate between predicting the next action $\hat{a}_{t+i} = f_\pi(\{s_{t-k}, a_{t-k}, \cdots, s_t, \hat{a}_t, \cdots, \hat{s}_{t+i}\}, z^\pi)$ and the next state, reward, and return
$\hat{s}_{t+i+1}, \hat{r}_{t+i}, \hat{R}_{t+i+1} = f_w(\{s_{t-k}, a_{t-k}, \cdots, s_t, \hat{a}_t, \cdots, \hat{s}_{t+i}, \hat{a}_{t+i}\}, z^w)$. We repeat this alternating procedure until we reach the horizon length h and compute $\hat{\tau}_t^h$ and its corresponding $\hat{R}(\hat{\tau}_t^h)$.

Because we use discrete latent variables, we can enumerate all possible combinations of z^π and z^w. There are c^{n_π} possible values for z^π and c^{n_w} possible values for z^w, which leads to $c^{n_\pi + n_w}$ different possible candidate trajectories. In this work, we found $c = 2$, $n_w \leq 4$, and $n_\pi \leq 4$ to be sufficient for all our explored problems. Thus, we only need to consider a maximum of 256 different combinations of latent variables, which is a standard batch size in many deep learning applications. Therefore, it is easy to run the candidate trajectory generation procedure previously described for all combinations of latent variables on modern GPU hardware.

Without loss of generality we can order the possible values of z^π and assign each one an index $i \in [1, c^{n_\pi}]$. We can do the same for z^w and assign each one an index $j \in [1, c^{n_w}]$. Then, we will label the candidate trajectories that are produced when conditioned on the ith z^π and jth z^w as $\hat{\tau}_{ij}$ and its corresponding return $\hat{R}_{ij}$. Then, we select the candidate trajectory that corresponds to $\max_i \min_j \hat{R}_{ij}$. We execute the first action of $\hat{\tau}_{i^* j^*}$ and repeat this procedure at every timestep. The intuition behind this procedure, is that we are trying to pick a policy to follow that will be robust to any realistic possible future in the current environment. Later, we will show how this procedure allows our method to be opportunistic in safe situations and cautious in more dangerous situations.

3.5 Experiments

For all experiments, we compare our SPLT Transformer method to Trajectory Transformer (TT), Decision Transformer (DT), and Behavioral Cloning (BC) with a Transformer model.

3.5.1 Illustrative Example

We start with a toy autonomous driving problem that we designed to be very simple, but that still demonstrates the dangerous optimism bias in prior Transformer-based approaches.

Metric	**SPLT (Ours)**	**BC**	**DT(m)**	**DT(e)**	**TT**	**TT(a)**
Return	**78.5** ± 0.2	58.0 ± 1.0	28.5 ± 0.4	68.9 ± 0.4	58.8 ± 0.5	67.3 ± 5.6
Success(%)	**100.0** ± 0.0	**100.0** ± 0.0	50.0 ± 0.0	**100.0** ± 0.0	**100.0** ± 0.0	91.7 ± 5.8

Table 3.1: We evaluate all methods with 3 seeds and on 100 different trials in the environment. We report the mean and standard deviation across seeds. DT(m) is DT conditioned on the maximum return in the dataset. DT(e) is DT conditioned on the expected return of the best controller used to collect the dataset. TT(a) is TT with more aggressive search parameters. For reference, the best IDM controller in the distribution of controllers used to collect the data gets a return of 78.6. We bold the highest mean.

In this toy problem, we use our own simple simulation environment where we have an ego vehicle trailing a leading vehicle with both travelling in the same direction on a 1-D path. Both vehicles are represented using point-mass dynamics, but only the ego vehicle is controllable. Half of the time the leading vehicle will begin hard-braking at the last possible moment in order to stop just before the 70m mark before continuing. The other half of the time the leading vehicle will immediately speed up to the maximum speed and continue for the entire trajectory. The ego vehicle cannot infer beforehand whether the leading vehicle will brake or not, and thus this is a completely stochastic event from the perspective of the ego vehicle.

The observation space is the absolute position and velocity of both the ego and leading vehicle. The action space is just the acceleration for the ego vehicle clipped to $[-1, 1] m/s^2$. The maximum velocity for both vehicles is $10 m/s$ and the minimum velocity is $0 m/s$, so the vehicles cannot travel backwards. The ego vehicle is rewarded for the distance traveled at each timestep, but is given a penalty of -100 if it crashes. The trajectory ends after $10s$ or if the ego vehicle crashes into the leading vehicle. The ego vehicle is initialized at $0m$ with a random velocity in $[7.5, 10] m/s$. The leading vehicle is initialized randomly within $[10, 20] m$ and with the same velocity as the ego vehicle.

For the offline dataset, we collected ~ 100000 steps with a distribution of different IDM [109] controllers that demonstrate a wide range of aggressiveness, and includes some trajectories where the IDM controller is too aggressive and collides with the leading vehicle. We show our results in Table 3.1.

For DT, we find that conditioning on the maximum return in the dataset (DT(m)) leads to crashes every time the leading vehicle brakes. If we condition on the mean return of the best controller used to collect the dataset (DT(e)), then we get the opposite behavior. The agent does not crash, but also does not take full advantage of the situations where the leading vehicle does not brake, and thus underperforms. While we were able to tune the conditional return in order to get a reasonable return result of 73.4, we found the optimal value to be quite arbitrary. Thus, we believe this parameter will be very difficult to tune in more general and complex stochastic environments, where the possible returns would be very hard to estimate beforehand without significant prior knowledge of the specific testing scenario.

For TT, we find that the results depend heavily on the scope of the search used. When we use the default parameters from their codebase for the beam search (TT), we get results very comparable to behavior cloning, which is quite suboptimal. When we reduce the low-probability filtering to allow for more aggressive search (TT(a)), we find that the method sometimes crashes into the leading vehicle because it picks the predicted trajectory where both vehicles will continue at max speed. Similar to DT, we expect tuning the search aggressiveness for TT to be difficult without significant prior knowledge of the intended testing scenario.

For our method, we find that our world VAE is able to predict both possible modes for the leading vehicle's behavior. Additionally, the policy VAE seems to be able to predict a range of different trailing behaviors. Thus, our method is able to properly search for an effective and robust behavior and achieves results comparable to the best IDM controller in the distribution used to collect the data.

3.5.2 NoCrash

Next, we evaluate our method on the CARLA [31] NoCrash [22] benchmark. For these experiments, we run the 0.9.11 version of CARLA at 5fps. We assume access to a global route planner that can generate dense waypoints to our goal, an accurate localization system, and a perception system that can identify the state of any vehicle or traffic light directly in front of us within a limited sensing range. In our experiments, we obtain these ground truth quantities from the CARLA simulator, as commercial self-driving car efforts already have systems to provide these quantities. Thus, we leave implementing such systems as beyond the scope of this project.

The goal in the CARLA NoCrash benchmark is to navigate in a suburban town to a desired goal waypoint from a predetermined start waypoint. The benchmark takes place in the CARLA towns Town01 and Town02 and consists of 25 different routes in each town. For our observation we use a low-dimensional vector representation consisting of: **(1)** the relative heading error to the next target waypoint, **(2)** the distance from the center of the target lane, **(3)** the ego vehicle speed, **(4)** the relative distance to the leading vehicle or the max sensing range if there is no leading vehicle in range, **(5)** the speed of the leading vehicle or the max speed if there is no leading vehicle in range, **(6)** the distance to the upcoming red light or the max sensing range if there is no red light in range. There are 2 actions: the steering and the target velocity for a PID controller. The ego car is rewarded for traveling faster and receives a small penalty for deviating from the target lane and a large penalty for crashing or incurring a traffic infraction. We terminate the trajectory whenever the car crashes, incurs an infraction, times out, or reaches the goal.

We collect our offline dataset with autopilot agents with a distribution of different levels of aggressiveness. This aggressiveness corresponds to the parameters of a time-to-collision-based controller that is used to adjust the vehicle's speed in response to any leading vehicles. Additionally, the autopilots use a PID controller for steering, and always immediately brake if they are too close to a red light. We collect $\sim$300000 steps with these autopilots in the Town01 routes in the dense traffic setting. Due to the variations in aggressiveness, these autopilots demonstrate a wide range of imperfect behaviors like crashing into other vehicles or driving unnecessarily slow during data collection.

Metric	**SPLT (Ours)**	**BC**	**TT**	**DT(m)**	**DT(t)**	**AP(t)**
Success (%)	**99.7** $\pm$ 0.5	97.7 $\pm$ 1.9	87.0 $\pm$ 4.9	93.0 $\pm$ 6.4	96.0 $\pm$ 3.7	100.0
Speed (m/s)	2.79 $\pm$ 0.08	2.74 $\pm$ 0.09	2.85 $\pm$ 0.06	2.93 $\pm$ 0.04	**2.96** $\pm$ 0.04	2.75

Table 3.2: We evaluate all methods with 3 seeds and on 4 different runs through all 25 routes in the unseen Town02. We report the mean and standard deviation across seeds and bold the learning-based approach with highest mean. DT(m) is Decision Transformer conditioned on the maximum return in the dataset. DT(t) is DT with a hand tuned conditional return. AP(t) is the best autopilot controller from the distribution used to collect the data.

We evaluate all methods by training on this Town01 dataset and then running in the unseen Town02 routes with the dense traffic setting. We depict our results in Table 3.2.

We find that TT and DT conditioned on the max return (DT(m)) have a lower success rate than BC, which we suspect is due to the optimism bias we previously described leading to unnecessary collisions and infractions. When we tune the target return for DT(t) it can outperform BC in terms of average success rate and speed. However, similar to the toy problem, we found the tuned return to be quite arbitrary. Without online evaluation or prior knowledge of the testing domain, it would be quite difficult to estimate the best target return, especially considering that the testing scenarios are different from the training scenarios. Our SPLT method achieves a higher average success rate than all of our learning-based baselines, while still maintaing a higher average speed compared to BC. We suspect that our positive results are due to our method's planning procedure, which avoids the optimism bias of TT's naive beam search.

3.5.3 Leaderboard

Metric	**SPLT (Ours)**	**BC**	**DT(m)**	**DT(t)**	**TT**	**AP**
DS	**52.4** $\pm$ 5.8	46.4 $\pm$ 6.9	41.8 $\pm$ 4.8	51.0 $\pm$ 12.5	42.3 $\pm$ 11.7	84.6 $\pm$ 1.5
RC (%)	90.6 $\pm$ 8.0	**91.0** $\pm$ 9.4	73.6 $\pm$ 5.9	84.1 $\pm$ 11.0	66.4 $\pm$ 9.0	99.6 $\pm$ 0.3

Table 3.3: We train and evaluate each method using 3 seeds on the public Leaderboard testing routes. We report the mean and standard deviation across seeds. Driving scores (DS) are calculated using the official Leaderboard evaluator. For both driving score and route completion (RC) (%) a larger value is better. DT(m) is Decision Transformer conditioned on the maximum return in the dataset. DT(t) is DT with a hand tuned conditional return. AP is the autopilot controller we used to collect the dataset.

Next, we evaluate our method on a modified version of the CARLA Leaderboard[3] benchmark. For these experiments, we run the 0.9.10.1 version of CARLA at 10fps. The only additional assumption we make is that our perception system can identify the state of any vehicles or pedestrians directly surrounding us in all directions within a limited sensing range.

The CARLA Leaderboard benchmark is much more comprehensive and requires the agent to perform more involved maneuvers, like lane-changing in urban and highway situations. The major difference from our NoCrash setup is that we introduce 8 additional variables to the observation, corresponding to the distance and speed of surrounding vehicles in each of the 4 diagonal directions. We collect $\sim$1.2 million time steps using the

[3]https://leaderboard.carla.org/

autopilot from the Transfuser [83] codebase[4] in the CARLA Challenge 2021 training routes for our offline dataset. We evaluate all methods on the publicly available testing routes. We depict our results in Table 3.3.

Our method achieves the best overall driving score among all of our learning baselines. Driving score is a comprehensive indicator for driving quality that accounts for route completion, collisions, and traffic infractions. The increased complexity in the Leaderboard scenarios leads to the world being less predictable and cooperative from the ego-vehicle's perspective. Thus, our SPLT Transformer method, which disentangles the world dynamics and the agent decision-making process, is better equipped to handle this stochastic and uncooperative environment. We believe that TT in particular underperforms relative to our SPLT method because its naive beam search does not plan appropriately for the range of possibly uncooperative multimodal outcomes.

3.6 Conclusion

In this chapter, we presented our SeParated Latent Trajectory Transformer (SPLT Transformer) method, which trains two separate policy and world VAE models that can be used at test-time to efficiently perform robust search. We discussed how our approach avoids the optimism bias that other Transformer-based approaches for offline RL struggle with in stochastic settings. Finally, we demonstrated how our method outperforms these baseline approaches on a variety of autonomous driving tasks.

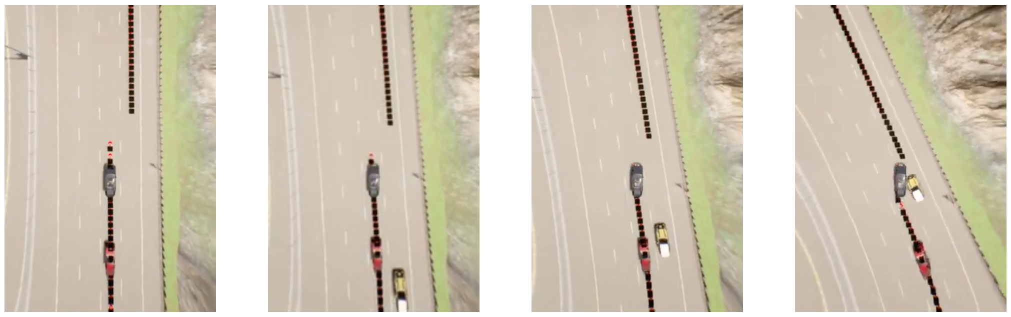


Figure 3.3: Visualization of a failed lane change in CARLA during our Leaderboard experiments. The black car is the ego-vehicle, and it is tasked with following the route indicated with black squares. The black ego-vehicle initiates a lane change despite the faster trailing yellow vehicle in the target lane, which leads to a collision.

[4]https://github.com/autonomousvision/transfuser

While our approach does achieve the best driving score among our different learning-based approaches, it still underperforms by a large margin compared to the autopilot used to collect the dataset. Empirically, we find that many of the failures of our SPLT approach come from interactions with vehicles that are approaching from the side or from behind in a different lane, which is particularly relevant when lane changing or navigating intersections. These failures seem to be caused by the limitations of our observation space and model representation. With the current observation space, it is almost impossible for the ego-vehicle to anticipate a vehicle approaching from these directions, especially if the approaching vehicle is traveling at a significantly different speed from the ego-vehicle, as illustrated in Figure 3.3. Thus, in the next chapter we will discuss how we can incorporate recent advancements in multi-agent traffic forecasting models to better handle these different types of interactive scenarios.

CHAPTER 4

Tractable Joint Prediction and Planning over Discrete Behavior Modes for Urban Driving

In order to better model the complex multi-agent interactions in real-world traffic, we move away from abstract feature spaces and instead shift towards entity-centric representations that are explicitly aware of the various other relevant entities in the scene like vehicles, pedestrians, and road elements. We take inspiration from many recent works in motion forecasting [76; 77; 38] and use a Transformer [112] neural network to process and learn the complex interactions between all these various entities in the scene. While these types of models have been used to achieve impressive results in multimodal trajectory forecasting, effectively integrating these models with downstream planners and model-based control approaches is still an open problem. Although these models have conventionally been evaluated for open-loop prediction, we show that they can be used to parameterize autoregressive closed-loop models without retraining. We consider recent trajectory prediction approaches which leverage learned anchor embeddings to predict multiple trajectories, finding that these anchor embeddings can parameterize discrete and distinct modes representing high-level driving behaviors. In this chapter, we present an approach that performs fully reactive closed-loop planning over these discrete latent modes, allowing us to tractably model the causal interactions between agents at each step. We validate our approach on a suite of more dynamic merging scenarios, finding that our approach avoids the "frozen robot problem" which is pervasive in conventional planners.

4.1 Introduction

In dense interactive situations like exiting a crowded parking lot after a sports game or merging during rush hour traffic, we require a robust model of how other agents might behave, and crucially, how they might respond to our own actions. To this end, significant progress has been made towards learning trajectory forecasting models from large datasets of urban driving logs [106; 111; 77; 38; 11]. State-of-the-art trajectory prediction models can capture the highly stochastic and multimodal distribution of outcomes in driving without needing to manually engineer complex human driving behaviors. In this work, we aim to explore fully leveraging these learned models for model-based planning and control.

Planning and prediction are usually treated as separate modules within a conventional autonomy stack. Usually a forecasting model will make predictions for all of the actors in the scene, and then plan open-loop against a set of possible open-loop trajectories for other agents in the scene. While this may ensure collision avoidance, it can cause the agent to behave overly conservatively, resulting in the "frozen robot problem" — if we are highly uncertain about nearby actors, then the only perfectly safe course of action is to do nothing, which can result in deadlock. Ideally, we should perform fully reactive closed-loop prediction and planning, such that the ego-vehicle actions directly affect the predicted behaviors of other agents, and vice versa.

Unfortunately, performing fully reactive closed-loop planning over a learned forecasting model is often computationally intractable. The distribution over agent trajectories is highly multimodal and subject to change at every timestep, which means the space of possible outcomes grows exponentially over time. It is also impossible to enumerate and check every possible future, since the distribution is continuous. Existing work [92] performs fully reactive closed-loop planning by first parameterizing agent policies as continuous latents in a normalizing flow [88], and then optimizing the ego-agent policy with respect to some differentiable cost function, but these approaches have previously only been shown to work at small scale with one or two other interacting agents. We show later that our approach scales favorably on more challenging tasks.

In this chapter, we devise a novel approach for performing fully reactive closed-loop planning over multimodal trajectory prediction models. We consider recent approaches to trajectory prediction that leverage learned anchor embeddings [111; 38; 77] to predict a diverse set of trajectories. While these approaches have previously only been evaluated for open-loop prediction, to the best of our knowledge ours is the first work to show how these models can also be used for closed-loop planning for dense urban driving. Our contributions are as follows:

- **Closed-loop prediction with open-loop training.** We show how trajectory prediction models trained open-loop with learned anchor embeddings can be used to parameterize autoregressive closed-loop models without the need for retraining.

- **Fully reactive closed-loop planning over discrete latent modes.** We propose a novel planning approach which leverages discrete latent modes to do planning over a compact latent behavior space. Our planning approach scales linearly with the number of agents and the planning horizon, as opposed to naive search, which scales exponentially. This allows us to tractably model the causal interactions between agents when performing autoregressive rollouts; the predictions for other agents react to the planned ego-trajectory and vice versa. Similar to [92], this allows us to perform fully reactive closed-loop planning, but over discrete rather than continuous latent conditioned policies. We find that our discrete formulation ensures diverse trajectory proposals and improves downstream planning performance.

We validate our approach called Predicting and Planning over Discrete Behavior Modes (P2DBM) on a challenging suite of highly interactive urban driving scenarios, outperforming the demonstrator agent and several strong baselines. Our P2DBM approach also beats the previous state-of-the-art in CARLA [31] on the Longest6 benchmark [83] when evaluated at realistic speeds.

4.2 Related Work

4.2.1 Trajectory forecasting models for driving

There has been a substantial amount of work towards training trajectory forecasting models for urban driving [91; 106; 11; 77; 38; 111]. However, none of these papers performs evaluations on closed-loop planning tasks beyond doing log-replay on offline driving logs for short time horizons. Most closely related to the model we use are Scene Transformer [77] and AutoBots [38], which both use vector-based input representations and large Transformer models. Our model can support many state-of-the-art forecasting approaches, and only requires minimal modification to most in order to facilitate planning. Note that although all of these approaches are trained to perform open-loop prediction, our P2DBM approach demonstrates how they can be adapted to perform fully reactive closed-loop prediction and planning without needing to substantially modify the training procedure.

4.2.2 Planning over learned forecasting models

There is a substantial amount of literature on planning over learned forecasting models for driving. Several prior works plan over imitation models, but do not perform closed-loop planning, instead optimizing a fixed ego-trajectory [90; 126; 103; 67; 48]. Another common approach is to not model the influence of the ego-agent on other agents [44; 127; 24]. As we will demonstrate, these approaches are unable to perform proactive planning when forced into close interactions with dense traffic. Model-based reinforcement learning approaches such as [119; 54] can theoretically leverage learned models and perform closed-loop prediction and planning for autonomous driving. However, many of these approaches have been restricted to small state spaces and have not been shown to perform well on more difficult closed-loop driving environments. In particular, we argue that rich vector-based state representations such as the one used in this work are key to effective prediction and planning. [49] models joint interactions between agents using a game-theoretic framework and evaluates planning performance in simulation, but does not use online fully reactive closed-loop planning like our approach.

The most similar work to ours in the literature is CfO [92], which also does fully reactive closed-loop planning for driving in CARLA. The tasks they consider are significantly simpler than the ones considered in this paper – they only consider interactions with one other vehicle (we consider up to 100). Their approach also relies on an autoregressive normalizing flow model which, to our knowledge, has not yet been scaled to larger-scale datasets, whereas our model architecture is similar to other Transformer-based forecasting models in the literature which have been deployed at scale.

4.2.3 Learning to drive in CARLA

Most competing approaches in CARLA [31] are based on imitation learning [14; 15; 13; 83; 18]. While most approaches focus on the visual imitation setting, our closest competitor is PlanT [87], which also does imitation learning, but operates in a similar setup to ours, where we assume the perception problem is solved. Imitation learning approaches rely on collecting optimal demonstrations, which can be challenging as we scale up to harder scenarios and more realistic data collection settings. We show that unlike imitation learning, our P2DBM method is able to out-perform the demonstrator on our suite of challenging urban navigation tasks.

4.3 Trajectory Prediction

4.3.1 Model formulation

In this section, we present our model architecture used for trajectory prediction. Let x_t^a denote the state of vehicle a at timestep t (we assume x_t^0 and $x_t^{1:A}$ are the vehicle states for the ego-vehicle and the rest of the vehicles respectively). We also assume that $t = 0$ is the current timestep and $x_{\leq T}$ refers to future vehicle states $x_1, ...x_T$ (ignoring the current timestep). Our goal is to model the distribution over future vehicle states $P(x_{\leq T}|x_0, c)$ where c is some arbitrary conditioning information (omitted from now on for brevity). We can factorize this distribution autoregressively over time, as is common in conventional trajectory prediction approaches. In this work we only model marginal (as opposed to joint) distributions over agents, so we can additionally factorize our distribution as

$$P(x_{\leq T}|x_0) = \prod_{t=0}^{T} \prod_{a=0}^{A} P(x_{t+1}^a | x_t) \tag{4.1}$$

We propose a simple model formulation which captures multimodality and is well-suited for downstream reactive planning. We represent the high-level multimodal behavior of the agents with categorical latent variables $z_t^a \in [1, ..., K]$ for each agent:

$$P(x_{\leq T}|x_0) = \prod_{t=0}^{T} \prod_{a=0}^{A} P(x_{t+1}^a | x_t, z_t^a) P(z_t^a | x_t) \tag{4.2}$$

Following Casas et al. [11], we implicitly characterize $P(x_{t+1}^a|x_t, z_t^a)$ using a deterministic mapping $x_{t+1}^a = f(x_t, z_t^a)$. Therefore, we only require our model to learn this 1-step prediction mapping $x_{t+1}^a = f(x_t, z_t^a)$, and estimate $P(z_t^a|x_t)$. In practice, we find that training our model to predict H steps open-loop into the future $x_{t+1:t+H}^a = f(x_t, z_t^a)$ is a useful auxiliary task that improves the training and generalization of the desired 1-step prediction mapping. Below we show how $f(x_t, z_t^a)$ and $P(z_t^a|x_t)$ can be parameterized by a Transformer model.

4.3.2 Network architecture

We design our architecture to predict each of the different modes for all agents independently, but all in one pass of the model. Thus, our model outputs a vector with shape $[K, A, H, 4]$ of potential future states of all agents in the scene, where K is the number of modes and A is the number of agents. This corresponds to $K \times A$ trajectories of length H, where each waypoint consists of position, heading, and speed. The high-level structure of our model is depicted in Figure 4.1.

Since the number of agents and context objects in the scene is not fixed, we adopt a flexible vector-based representation of the scene. We consider two types of entities: vehicles $\mathcal{X}$ and scene context objects $\mathcal{C}$ which consist of road points, traffic lights, pedestrians, stop

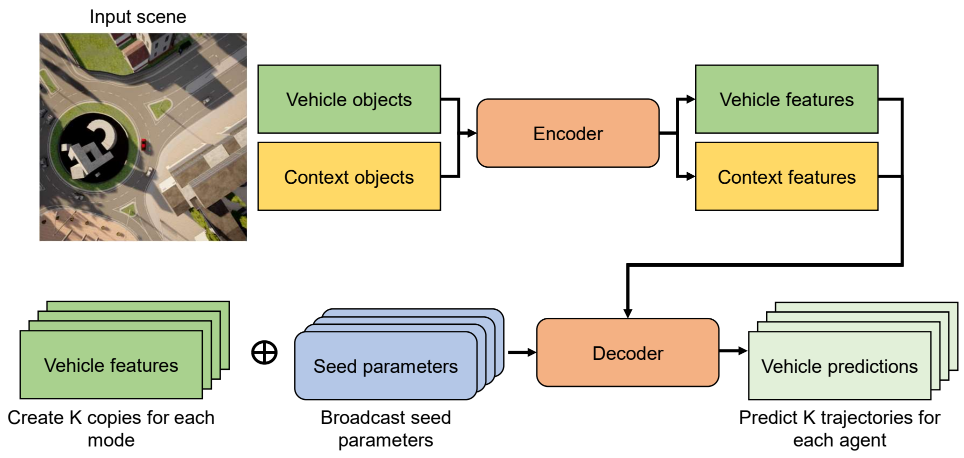


Figure 4.1: Forecasting model architecture.

signs, and goal waypoints. Each object is represented by a feature vector that contains the relevant raw information. For vehicles, this is relative pose, bounding box, speed, and current effective speed limit. For road points this is relative pose, lane width, whether the point is in an intersection, and whether one can change lanes to the left and right. For traffic lights this is relative pose, affected bounding box, and light state. For pedestrians this is relative pose, bounding box, and speed. For stop signs this is relative pose and affected bounding box. For goal waypoints this is relative pose and lane width. We assume access to all agents and context objects within 50m of the ego-vehicle up to a hard cap for each class of object. Each object is encoded using a MLP (one for each object type) to ensure all object features have the same dimension D. Then the entire set of input features is processed by an encoder to produce encoded vehicle features: $\mathcal{X}' = \text{Enc}(\mathcal{X}, \mathcal{C})$.

The encoder consists of stacked cross-attention layers where each vehicle cross-attends to both the vehicle features $\mathcal{X}$ and the context features $\mathcal{C}$ concatenated together. In practice, we find that only allowing each vehicle to attend to nearby map features improves performance, so we only attend to the closest 50 map features. Additionally, we use relative positional embeddings to maintain translational invariance. Prior work [25] has demonstrated that translational invariance is a useful inductive bias when performing multi-agent trajectory prediction. When feature u_i cross-attends to u_j, we compute query Q_{ij}, key K_{ij}, and value V_{ij} as follows:

$$\begin{aligned} Q_{ij} &= W^q u_i \\ K_{ij} &= W^k (u_j + p_{ij}) \\ V_{ij} &= W^v (u_j + p_{ij}) \end{aligned}$$

where W^q, W^k, W^v are learned projection matrices and p_{ij} is a learned relative positional embedding. All features have an associated spatial position (x, y) as well as an orientation θ. To compute p_{ij}, we compute the pose (x_j, y_j, θ_j) of feature u_j transformed to the frame of feature u_i at pose (x_i, y_i, θ_i) to get a relative pose $(x_{ij}, y_{ij}, \theta_{ij})$. We then construct a relative pose feature by concatenating x_{ij} and y_{ij}, as well as $\sin(\theta_{ij})$ and $\cos(\theta_{ij})$. Finally, an MLP is used to encode this relative pose feature to obtain p_{ij}. Note that we do not include any absolute positional information in our features, so the network is only able to observe the relative poses of scene objects.

Following the encoder, our encoded vehicle features are of shape $[A, D]$. We expand these features to be of shape $[K, A, D]$ so that each unique combination of mode and actor corresponds to a single D-dimensional feature. Following [38; 111; 77], we introduce learnable anchor embeddings $M \in \mathbb{R}^{K \times D}$ for each of the K modes, which are broadcasted and added to the encoded vehicle features in order to distinguish the different modes. We learn two separate sets of anchor embeddings: one for the ego-vehicle only, and one for all other vehicles.

For each combination of agent and mode we get the query features $Q_k^a = \mathcal{X}'^a + M_k$ with appropriate broadcasting. Then, we use the decoder to produce K trajectory predictions for each agent: $Y_k = \text{Dec}(Q_k^a, \mathcal{X}')$. The decoder consists of two alternating operations: the query features Q_k^a cross-attend to both the encoded vehicle features $\mathcal{X}'$ and map features $\mathcal{C}$, and then the query features Q_k^a self-attend along the K modes dimension. In other words, for each agent, the different mode features self-attend to each other. Note that during self-attention, only features corresponding to the same agent can attend to each other. This ensures that each agent future is decoded independently.

Finally, the processed query features are passed to a MLP. Since each unique agent and mode has its own feature, each D-dimensional feature is mapped to a trajectory of length H, where each waypoint consists of positions, orientations, and speeds. For each agent, we predict relative to that agent's coordinate frame, and also for each time step predict in-frame displacements rather than absolute positions, orientations, and speeds.

4.3.3 Training objectives

We train the model to predict the ground truth trajectories using a negative log-likelihood loss, where each prediction is parameterized by a Gaussian. Note that although we train our model to parameterize a Gaussian, we always take the mean when unrolling our model. In order to ensure our model can capture multimodal outcomes, we adopt a winner-takes-all objective [75] where only the closest of the K predictions will backpropagate its loss. Since we are doing marginal prediction, our winner-takes-all objective takes the best prediction for each agent separately. Our model is also trained to output a logit for each mode to estimate the probability of that mode being used, and this is trained using a cross-entropy loss with a one-hot target for the closest of the K predictions.

4.4 Closed-Loop Planning over Discrete Behavior Modes

4.4.1 Planning with autoregressive rollouts

Our objective is to find the behaviors for the ego-vehicle that will maximize the discounted sum of rewards over T timesteps, where γ is a discount factor: $\sum_t^T \gamma^t R(x_t, c)$. Because the ego-vehicle's decision in the trajectory is determined by the chosen z^0, we can write our objective as

$$\underset{z^0}{\operatorname{argmax}}\, \mathbb{E}[\sum_t^T \gamma^t R(x_t, c)] \tag{4.3}$$

As illustrated in equation 4.2, we can sample $x_{\leq T}$ autoregressively using our Transformer model. For each timestep, we just deterministically decode $x_{t+1}^a = f(x_t, z_t^a)$ for our latent sample z_t^a.

Given our specific Transformer architecture, we can perform each autoregressive step of this procedure with just one forward pass of the model. We generate all the multimodal predictions at once by leveraging the anchor embeddings and pick the mode prediction that corresponds to $x_{t+1}^a = f(x_t, z_t^a)$. Note that even though we trained these models to make multi-step open-loop predictions, we only need to take the first step of the prediction, which is x_{t+1}^a. Then, we can repeat this procedure up to the decided horizon T in order to generate a sample of $x_{\leq T}$.

4.4.2 Evaluating ego-modes

Now in order to perform planning, we need to evaluate Equation 4.3 for specific values of z^0. Since the ego latents $z^0 = [z^0_0, ...z^0_{T-1}]$ can vary between timesteps, z^0 can take on K^T possible values (K possible ego-modes for T timesteps), which is computationally expensive to fully enumerate.

Empirically, we find that keeping the latent modes consistent across time (i.e. $z_0 = ... = z_{T-1}$) for both the ego-vehicle and surrounding vehicles leads to good performance, and thus we use this scheme in our experiments to reduce complexity. We hypothesize that this works because the learned anchor embeddings encourage the model to learn locally consistent modes since the embeddings are shared globally, i.e., they are not state-conditioned. Additionally, we find that conditioning on these different modes leads to meaningfully diverse behaviors over time. Thus by keeping the latent modes consistent across time, we only need to compare K ego latent modes to optimize Equation 4.3.

Specifically, we evaluate Equation 4.3 for any specific z^0 by generating N samples of $x_{\leq T}$ with the previously described procedure. At the first time step, for each surrounding agent we sample $z^a_0 \sim P(\cdot|x_0)$ and for the ego-agent we set z^0_0 to the ego mode we are evaluating. Then, we continue to set $z_t = z_0$ at every subsequent timestep during autoregressive generation. We estimate $\mathbb{E}[\sum_t^T \gamma^t R(x_t, c)]$ by taking the mean over these N samples. We do this for all K ego modes for z^0, and pick the mode with the maximum expected sum of rewards. We use MPC and execute the first step by using it as a target for a PID controller, and continue to replan at every step.

This approach enables us to perform fully reactive closed-loop planning. Note that we are planning over latent modes rather than open-loop trajectories, which means we can effectively model the causal influence of the ego on other vehicles and vice versa. Additionally, this allows us to be robust to different responses from other agents, since we can evaluate each candidate ego latent z^0 under a variable number of latent samples.

4.4.3 Reward function

Our reward function is as follows:

$$\begin{aligned}
R(x_t, c) &= \beta_{coll} R_{coll}(x_t, c) + \beta_{lane} R_{lane}(x_t, c) \\
&\quad + \beta_{speed} R_{speed}(x_t, c) + \beta_{light} R_{light}(x_t, c) \\
R_{coll}(x_t, c) &= -1_{coll} \\
R_{lane}(x_t, c) &= 1 - \frac{|\text{lateral}(x_t, c)|}{0.5 \times \text{(lane width)}} \\
R_{speed}(x_t, c) &= 1 - \frac{|\text{speed}(x_t) - \text{(speed limit)}|}{\text{(speed limit)}} \\
R_{light}(x_t, c) &= -1_{red} \frac{\text{speed}(x_t)}{\text{speed limit}}
\end{aligned}$$

where β_{coll}, β_{lane}, β_{speed}, and β_{light} control the relative weight assigned to each reward term. The collision term R_{coll} is -1 and leads to a termination if the ego collides with another object and 0 otherwise. The lane term R_{lane} penalizes deviation from the route, where lateral(x_t, c) is the lateral distance of the ego from the closest route segment. The speed term R_{speed} encourages the ego to drive at the speed limit. The light term R_{light} penalizes the ego for having non-zero speed if the light is red; the penalty is scaled by the driving speed proportional to the speed limit.

4.5 Results

4.5.1 Model training and hyperparameters

For all experiments, we train our method with 4 encoder layers consisting of a cross-attention layer followed by a feedforward layer, and 4 decoder layers consisting of a cross-attention layer, self-attention layer, and a feedforward layer. Following [122], we use Pre-Layer Normalization. For the merging scenarios, we use an embedding size of 128 for a model with approximately 1.9 million parameters. For the CARLA Longest6 benchmark, we use an embedding size of 256 for a model with approximately 7.4 million parameters. We train our model with AdamW [69; 56] with an initial learning rate of $2e-4$ with cosine annealing to 0 as we train for 100 epochs. We train our multimodal model with $K = 8$ latent modes and use $N = 8$ samples to estimate expected rewards during planning.

For both experiment settings, we use a visualization range of 50m and observe the closest vehicles by distance up to a cap of 100 vehicles. For the merging scenarios, we set

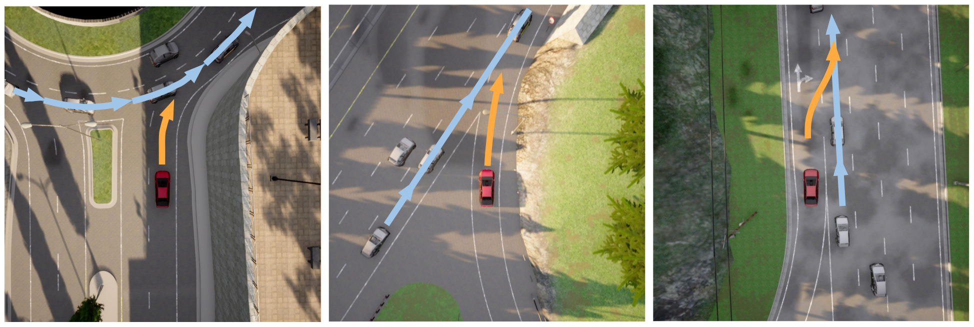


Figure 4.2: Merge scenario visualization. The red car is the ego-vehicle, the orange arrow indicates the desired merging behavior, and the blue arrows indicate the flow of traffic in the target lane.

the coefficients of the cost function as $\beta_{coll} = 20$, $\beta_{lane} = 0.1$, $\beta_{speed} = 1$, and $\beta_{light} = 0$, as there are no traffic lights in these scenarios. For the CARLA Longest6 benchmark, we set the coefficients of the cost function as $\beta_{coll} = 20$, $\beta_{lane} = 1$, $\beta_{speed} = 1$, and $\beta_{light} = 4$.

4.5.2 Merging scenarios

We design a suite of challenging urban navigation scenarios in order to evaluate whether our P2DBM approach can safely perform proactive maneuvers in dense traffic. We identify 10 distinct merging scenarios that each involve a high degree of interaction with other cars in the midst of dense urban traffic. These other cars are controlled by CARLA's internal traffic manager, which uses an adaptive controller with randomly initialized minimum safety distances and target speeds. In each scenario, the goal is point-to-point navigation where the goals are specified a priori. Three of these scenarios are visualized in Figure 4.2.

An episode is considered a success if the ego-vehicle reaches the goal without committing any traffic infractions; otherwise, the episode is considered a failure. A common theme we found in prior CARLA driving benchmarks was that they rarely assessed the ego-agent's assertiveness, instead relying solely on metrics related to collisions and other traffic infractions. As such, many of the top-performing approaches in CARLA tend to engage in overly conservative and unrealistic behaviors. To additionally assess the ability of each approach to be proactive, we introduce an additional infraction type for static episodes, i.e., situations where the car is stopped for long periods of time.

For each approach, we evaluate it on all 10 merging scenarios, each with 20 different random traffic initializations for a total of 200 episodes. We separately report success rates,

Approach	Success (%) $\uparrow$	Static (%) $\downarrow$	Crash (%) $\downarrow$
P2DBM (Ours)	**96.5** $\pm$ **1.8**	2.2 $\pm$ 1.0	**1.2** $\pm$ **0.8**
Open-Loop	78.3 $\pm$ 2.1	20.5 $\pm$ 2.3	**1.2** $\pm$ **0.2**
Multimodal IL	50.2 $\pm$ 1.0	46.5 $\pm$ 1.5	3.3 $\pm$ 0.6
Unimodal IL	51.8 $\pm$ 1.6	46.3 $\pm$ 2.	1.8 $\pm$ 1.0
Continuous Latents	71.8 $\pm$ 0.8	24.3 $\pm$ 1.4	3.8 $\pm$ 0.6
PlanT	67.2 $\pm$ 1.2	30.7 $\pm$ 0.9	2.2 $\pm$ 0.3
Data Policy	68.0	12.5	19.5
Autopilot	85.5	**1.5**	13.0

Table 4.1: We report results with averages and standard errors over 3 different seeds of running the 200 different episodes. $\uparrow$ and $\downarrow$ indicates that higher or lower numbers are better respectively. We bold the results with the best mean.

static rates, and crash rates for each approach. On these merge scenarios, we compare to several baselines:

- *Multimodal imitation*. Same Transformer model as our approach, except we execute the mode prediction with the highest predicted probability.
- *Unimodal imitation*. Same Transformer backbone as our approach, except we only use one mode and only make one prediction.
- *Ours w/ open-loop planning*. Same model as our proposed approach, except instead of unrolling our model autoregressively, we run our model once to generate marginal open-loop predictions for all cars. The planning procedure is the same as in the closed-loop case, except the trajectory predictions are fixed.
- *Ours w/ continuous latent variables*. We train a CVAE [102] with Gaussian latent variables with our Transformer backbone and do sampling-based planning for all agents (including the ego). The number of samples is equivalent to the total number of mode combinations expanded by our approach in order to ensure fair comparison.
- *PlanT*. We compare to PlanT [87], which is an imitation learning approach that is state-of-the-art on the CARLA Longest6 benchmark. We use the original code implementation of the model.
- *Data policy*. The configurable heuristic policy used to collect the dataset. We sample random configuration parameters (e.g. planning horizon, bounding box sizes, etc.) to generate diverse demonstrations.

- *Autopilot policy*. The best-performing configuration of the data collection policy.

To train all approaches, we collect a dataset of 100K samples using the data policy. We train our models on all scenarios and share them across baselines where possible. We report mean and standard error across 3 different model seeds.

Table 4.1 shows the results on the merge scenario suite. Our P2DBM approach outperforms the baselines and achieves the highest overall success rate. Notably our method is the only learned approach which actually outperforms the best-performing heuristic policy (autopilot). The open-loop variant of our approach performs worse than our closed-loop approach and has a higher static rate. The open-loop planner cannot predict that other vehicles will alter their current trajectory if the ego attempts to merge and therefore behaves much more conservatively, resulting in more static episodes. On the other hand, our closed-loop planner will predict that other vehicles will sometimes slow down to allow the ego-vehicle to merge. Figure 4.3 shows a qualitative comparison between the open and closed-loop planners. The imitation policies all have low crash rates due to excessive static episodes, and generally engage in extremely conservative behavior. This also suggests that the dominant modes in the dataset are highly cautious, even when we explicitly model multiple modes. However, our P2DBM approach is able to identify and execute the high-performance modes in the dataset.

Since our choice of discrete latents distinguishes our approach from previous work [92], we provide a more detailed comparison here between our approach and a CVAE baseline which uses continuous latents based on a Gaussian prior. We suspect the main reason for the disparity in performance is that our approach is better able to predict and sample distinct trajectories. Empirically we find that our discrete latent modes correspond to semantically different predictions that lead to a greater diversity of closed-loop predictions for both the ego-vehicle and the surrounding vehicles. On the other hand, taking arbitrary samples from a CVAE will not necessarily cover the relevant behavior modes, especially in a limited computation setting. Additionally, our approach explicitly enumerates distinct latent modes for the ego-vehicle independent of the predicted likelihood under the dataset, so it can more easily capture less common modes in the data.

4.5.3 Longest6 benchmark scenarios

We also evaluate our approach on the CARLA Longest6 benchmark, which consists of driving on a difficult subset of the publicly available CARLA Leaderboard routes with added dense traffic. Unlike the previous scenario suite, these routes are much longer and

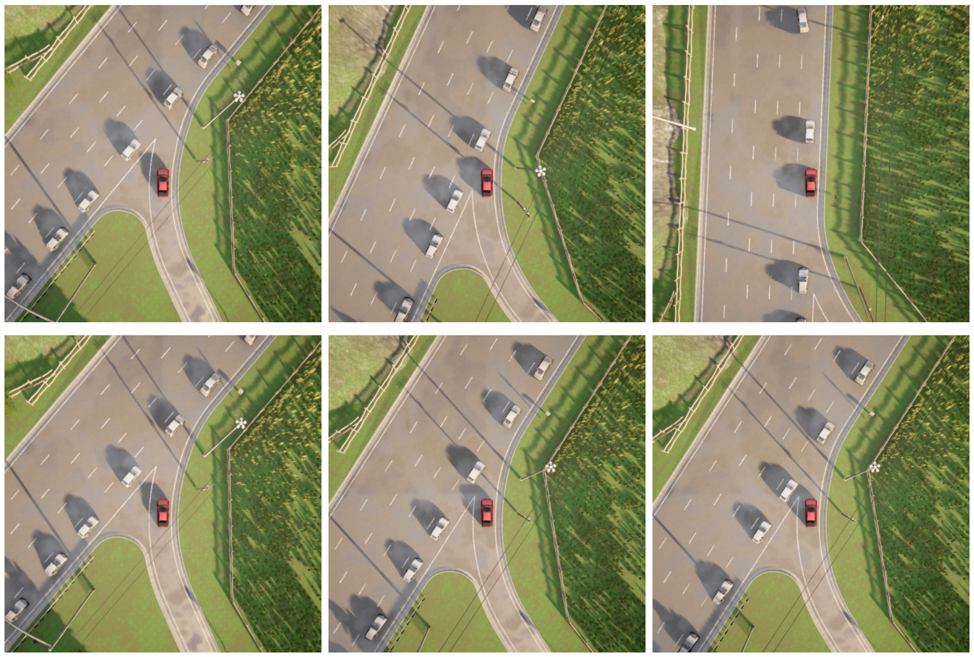


Figure 4.3: Qualitative example of proactive merging with closed-loop planning. The top row is from our proposed closed-loop planner and the bottom row is from the open-loop variant of our planner. Frames are in sequential order from left to right. The closed-loop planner merges proactively in front of other cars, causing the car behind to yield to the ego-agent. The open-loop planner does cannot predict that the ego will affect the behavior of other cars, so it does not attempt to merge.

consist of multiple scenarios defined according to the NHTSA typology.

We found that existing state-of-the-art approaches in CARLA Leaderboard are able to avoid many of the highly interactive scenarios by simply *driving significantly under the speed limit*. For instance, pedestrians on CARLA Leaderboard are regularly spawned to cross the road in front of the ego-vehicle at a constant speed. Many approaches avoid these scenarios by driving at a low enough speed such that the ego-vehicle will never be going fast enough to hit the spawned pedestrians. In general, driving at low speeds substantially reduces the degree of interaction present in these scenarios.

To circumvent this loophole, we collect demonstration data with an autopilot policy driving at the speed limit. Specifically, we use the same autopilot to collect the data that was used in both Prakash et al. [83] and Renz et al. [87], but have the target speed be the current speed limit instead of a constant 4 m/s. This significantly increases the level of

Approach	DS ↑	RC ↑	IP ↑
P2DBM (Ours)	**51.0** ± **1.2**	**89.2** ± **1.7**	0.582 ± 0.010
Autopilot	44.6 ± 1.6	84.0 ± 1.1	0.546 ± 0.040
PlanT	37.1 ± 1.4	86.1 ± 1.5	0.457 ± 0.017
Multimodal IL	48.4 ± 2.6	83.3 ± 2.1	**0.603** ± **0.021**

Table 4.2: We report results with averages and standard errors over 5 different seeds of running the 36 different routes. We bold the results with the best mean. Route Completion (RC) is the percentage of the route completed. Infraction Penalty (IP) is a multiplier that gets lower with each infraction or collision. Driving scores (DS) are calculated by multiplying the RC score and IP score for each route. Higher is better for all of these metrics.

interaction required to solve these routes. In order to facilitate fair comparison with existing approaches, we retrained a state-of-the-art approach PlanT [87] using our new dataset.

We adopt the main metric used by the CARLA Leaderboard. Driving score (DS) is the primary metric used to compare and rank approaches in CARLA Leaderboard, and is the product of a route completion score (RC) and an infraction penalty (IP) score. Route completion is the percentage of the route that is completed. Infraction penalty is a multiplier which penalizes various infractions such as collisions, running red lights, etc.

For baselines, we compare to the autopilot policy, a multimodal imitation baseline trained using our Transformer backbone, and PlanT [87]. We collect 440K samples using the modified autopilot policy and use this dataset to train all approaches. For each approach, we evaluate on all 36 routes each with 5 different random seeds for a total of 180 episodes and we report the mean over all episodes and standard error over the 5 seeds.

Table 4.2 shows the results on the CARLA Longest6 scenarios evaluated at reasonable driving speeds. Our P2DBM approach outperforms the autopilot used to collect the dataset as well as PlanT, which is the current state-of-the-art on the original Longest6 benchmark. While the imitation learning baseline has a better infraction penalty score, it ultimately has a lower overall driving score because it often deadlocks earlier in the denser routes and avoids many of the challenging interactions. In contrast, our approach is able to act proactively and make much more progress in these dense scenarios and achieve a superior driving score.

4.6 Conclusion

In this chapter, we proposed a novel approach for performing closed-loop planning over multimodal trajectory prediction models. Our approach is able to excel on a challenging suite of dynamic merging scenarios that require proactive planning behaviors. Additionally, we show that our approach outperforms state-of-the-art approaches on CARLA Longest6 scenarios when evaluated at reasonable driving speeds.

CHAPTER 5

Learning Hierarchical Driving Policies with Offline Reactive Simulation

In the previous chapter, we found that our P2DBM method was able to perform effective online control by planning to select a behavior corresponding to one of our learned discrete latent modes. In particular, we found that P2DBM outperformed our multimodal imitation learning baseline, despite the fact that both methods were actually selecting over the same pre-trained latent modes. The positive results of our P2DBM method indicate that our approach of using learned anchor embeddings [76; 77; 38] to parameterize discrete and distinct behavior modes in combination with training with the winner-takes-all objective [75] is an effective means of learning a suitable set of candidate behaviors for online control. However, the poorer performance of our multimodal IL baseline indicates that using a classifier that is trained to maximize the likelihood of the behavioral data is significantly less effective at selecting the best of these behaviors when run online. Thus, in this chapter we are interested in exploring how we can use the principles from offline RL that we explored in Chapter 2, in order to learn a better policy for selecting over the pre-trained behavior modes offline. In particular, we reformulate our problem as a hierarchical RL problem, where we treat the learned discrete latent modes as a discrete set of observation-conditioned skills with the corresponding modes of our pre-trained traffic forecasting model acting as the low-level policies. Then in order to train our discrete high-level policy, we additionally use our multimodal traffic forecasting model as a world model to collect offline simulated trajectories with reactive traffic. Inspired by the success of our P2DBM method, we develop a policy learning algorithm that resembles distilling the outputs of P2DBM run on offline samples plus an additional behavioral constraint. We validate our approach in our CARLA [31] merging scenarios and find that our offline hierarchical learning approach is able to significantly reduce the performance gap with P2DBM, while reducing the planning latency by more than a factor of 10. Finally, we find that our approach is able to outperform the state of the art for a purely learning-based motion planning policy in nuPlan [40], when run in closed-loop simulation.

5.1 Introduction

Deep stochastic forecasting models [76; 111; 25] trained on large and diverse datasets have achieved impressive results in a variety of complex and extensive motion forecasting benchmarks [32; 12] for autonomous driving. In particular, there recently has been significant progress in improving both diversity and precision in forecasting complex scenes [77] with many potential yet distinct outcomes.

These forecasting models generally perform well on coverage metrics like minADE, but struggle on closed-loop metrics when executing their maximum-likelihood actions. Hybrid approaches [49; 121] that select over the same learned proposals using online planning often achieve greatly superior closed-loop performance. This indicates that generally these forecasting models can predict proposals that cover the expert behavior, but struggle to learn to predict the specific correct behavior to execute when trained purely from behavior-cloning. While these hybrid approaches can achieve stronger metrics, the additional planning also increases the latency for decision making. This is often not directly considered during simulation, but a longer latency could lead to consequential delays when executing online control in tight interactive scenarios or when traveling at high speed. Ideally, we would like to reduce latency and get similar metrics by just using one forward pass of our model without requiring the additional computational complexity caused by invoking an online planner.

Towards this end, we reformulate the problem of selecting over the discrete proposals provided by these multimodal forecasting models as a hierarchical reinforcement learning problem. Prior works in both reinforcement learning and autonomous driving have explored using offline datasets in order to pretrain a continuous [80; 114; 2; 35] or discrete [71; 27] set of skills. The goal in these skill learning approaches is to learn a relatively small library of temporally extended actions that lead to useful behaviors in the downstream task. Then for online control, we can learn a high-level policy that learns to just select the most effective of these skills to execute in the current state.

In autonomous driving, this discrete set of skills should ideally represent semantically meaningful yet distinct behaviors in the given state, like the vehicle asserting itself or yielding at a merge. Thus, leveraging these pre-trained skills should both accelerate and simplify

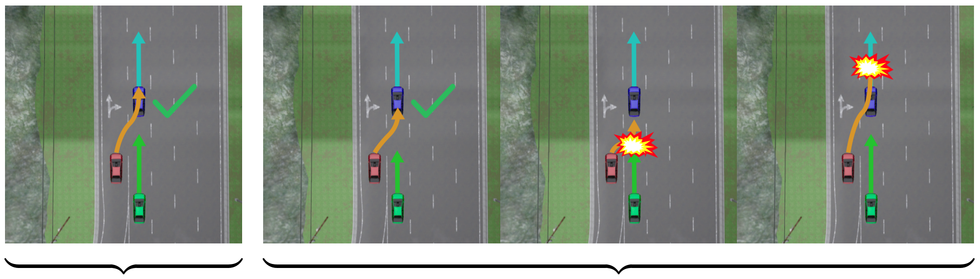


Figure 5.1: These images illustrate the red ego-vehicle attempting to merge from an on-ramp onto a highway. The leftmost image is the successful ground truth merge demonstrated in the logged dataset. The other three images illustrate the potential results of the ego-vehicle taking different counterfactual actions with the other vehicles reacting with closed-loop control. The second image illustrates that a slightly slower merge would still be successful because the trailing green vehicle should react and slow down for the ego-vehicle. However, the third image illustrates that merging too slowly could still lead to a collision if the green trailing vehicle is not given enough space to react. The final image illustrates that a faster merge could lead to a collision by rear-ending the blue leading vehicle. Better understanding the tolerances for a successful merge should help the ego-vehicle learn to generalize to harder merges with conditions that are slightly different than those observed in the logged data.

policy learning, as it should limit the search space to be over different high-level behaviors, rather than individual steering and acceleration commands. However, due to the safety-critical nature of autonomous driving, we would like to avoid the need for additional online samples in order to train our high-level policy. Therefore, we instead adopt an offline model-based approach similar to our prior work in Chapter 2, and use our pre-trained traffic forecasting model as a world model in order to collect offline samples to train our discrete high-level policy.

Unlike several prior works in using reinforcement learning to learn a driving policy offline [33; 70], we run the traffic forecasting model in closed-loop during the rollouts for the ego-vehicle. This will allow the other vehicles in the scene to dynamically adapt their behavior in potentially non-trivial ways in response to the ego-vehicle. With these dynamic offline simulations, we can explore counterfactual behaviors and learn from potential failure cases without needing to execute dangerous maneuvers in the real world (see Figure 5.1 for an example). Additionally, we can easily generate a diverse set of different responses from the other agents in the environment by taking different stochastic samples from our world model. This is important because from the perspective of the ego-vehicle, the other agents in the scene can act quite stochastically due to each agent's unknown intent, driving style, and alertness. Thus even for the same action sequence from the ego-vehicle, the same initial traffic scene can unfold in a variety of different ways. This inherent stochasticity means that any individual simulation or individual log in the dataset will often not be fully representative of the potential behaviors for other actors in the scene. Thus during policy training, we evaluate the different potential skills with respect to many different stochastic responses from the surrounding traffic in order to provide more accurate signal for training our high-level policy.

In particular, inspired by our results in Chapter 4, we develop an algorithm for learning the discrete high-level policy that is analogous to distilling the results of running a sample-based planner on the offline samples. Specifically, we enumerate and evaluate each of the discrete skills in each offline state by simulating several different closed-loop interactions with the surrounding traffic agents using our traffic forecasting model as a world model. Then, we simply distill the resulting rewards of running all of the different skills into our discrete high-level policy. Similar to many prior works in offline RL, we additionally adopt a soft behavioral constraint to prevent the high-level policy from diverging too far from the behavioral data. Our contributions are as follows:

- **Learning discrete skills with learned anchor embeddings.** We demonstrate that the approach used in many recent state-of-the-art traffic forecasting models [76; 77; 38] of learning multimodal predictions by conditioning on a discrete set of learned anchor embeddings to produce a discrete set of different trajectory decodings trained with a winner-takes-all training objective [75] can be an effective means of learning a discrete set of useful observation-conditioned skills.

- **Hierarchical policy learning with closed-loop offline trajectory simulation.** Furthermore, we propose a novel approach for offline learning a discrete high-level policy over these offline learned skills that involves using a pre-trained traffic forecasting model as a world model for simulating closed-loop rollouts with interactive traffic. In particular, our policy learning algorithm involves enumerating and evaluating all of the skills in each offline state over several stochastic samples, and distilling these results with an optional behavioral constraint into a discrete high-level policy.

We test our approach for Hierarchical Offline Learning with Offline reactive Simulation (HOLOS) in a curated set of merging scenarios run in the CARLA [31] simulator that require tight interaction and negotiation with stochastic actors. Finally, we validate the effectiveness of our HOLOS approach for learning a reactive policy by evaluating its closed-loop performance in the more extensive nuPlan [40] benchmark and demonstrate state-of-the-art results for a solely learning-based motion planning policy in closed-loop reactive simulation.

5.2 Related Works

5.2.1 Hierarchical Policy Learning

Many prior works have tackled learning for online control by learning a hierarchical policy where a high-level policy learns to select which low-level skills policy to execute in any given state. These low-level policies can either be hand-designed [28; 116] for the specific environment or be learned from data. Most relevant to this work, several prior works have explored extracting low-level skills from previously collected datasets, and using them to either facilitate online RL [80; 114; 99; 59] or to constrain offline model-free RL [2; 71]. In particular, Luo et al. [71] explored learning a discrete set of skills with a VQ-VAE [110] from offline data, and then using offline model-free RL to learn a high-level policy to pick over these discrete skills. In this work, instead of using these offline learned skills to accelerate online RL or as a constraint for offline model-free RL, we instead use offline simulated rollouts from our traffic forecasting world model in order to train a discrete high-level policy. Additionally instead of learning these discrete set of skills with a VQ-VAE [110] like Luo et al. [71], we instead use an approach that is inspired by recent advances in multimodal traffic forecasting models [76; 77; 38]. Specifically, we represent each of the discrete learned skills as the different trajectory decodings from our multimodal forecasting model, where each trajectory is decoded with a different learned anchor embedding and the whole multimodal model is trained with a winner-takes-all objective [75].

5.2.2 Imitation Learning in Autonomous Driving

Many prior works have explored using IL to learn a driving policy directly from sensors [8; 82] or from the outputs of a developed perception system [87]. Imitation learning directly from the driving logs will often suffer from distribution shift as the ego-vehicle visits states outside of the training distribution when deployed in real-world environments, which leads to accumulating errors and catastrophic failures. Several prior works [8; 3] have ameliorated some of these issues with heuristic-based data augmentations. While these data augmentations are quite effective at reducing the more trivial failure cases like driving off the road or swerving, they are less effective at improving the safety performance in the more challenging driving scenarios [70]. Additionally, several prior works have demonstrated that imitation learning can struggle due to causal confusion [30] or the “copy-cat problem” [117; 21], where the autonomous agent erroneously selects a behavior that is correlated with past actions rather than the current state. For example, in the dataset if the autonomous agent is currently stopped, then it quite likely that it will continue to stay still in the next time step. Thus, once the agent comes to a full stop during online execution, it can be biased towards staying stopped, even if the original cause for the stop is no longer present. In particular, De Haan et al. [30] demonstrated how causal confusion could be alleviated by evaluating a limited number of online samples in the environment in order to pick the most effective causal graph for a learned policy. In this work, instead of requiring online samples in the environment, we use offline simulated trajectories with our traffic forecasting world model in order to determine the most effective of our pre-trained skills and use that to learn a discrete high-level policy.

5.2.3 Multi-Agent Motion Forecasting

One of the major challenges in forecasting vehicles in autonomous driving is that in many situations surrounding vehicles will act stochastically with a multimodal distribution. For example, when merging with another vehicle, some drivers might yield and slow down, while other drivers will speed up and assert themselves depending on their driving style and the specific situation. Recent works have continually improved both the diversity and precision of these models by incorporating multiple learned anchor embeddings [111; 106; 38], and using Transformer [112] neural networks to handle the arbitrary number of agents in the scene. Depending on the intended use case of these models, some approaches model the marginal distribution for each agent independently [76], while other approaches model the joint distribution over all agents [77].

5.2.4 Learning for Planning in Autonomous Driving

In traditional autonomy stacks, multi-agent motion forecasting is used to instantiate occupancy maps for future timesteps, which is used to guide trajectory optimization. Several different hybrid approaches have explored incorporating forecasting models to generate initializations for trajectory optimization [47; 49] or to propose candidates for a sampling-based approach [49; 121]. Most of these approaches do evaluations with open-loop predictions for the ego-vehicle and for the other agents in the scene because these predictions can be made quickly with one forward pass of a forecasting model which facilitates online execution. In this work, because we are using our model to perform offline simulation for training our high-level skill selection policy, we can simulate rollouts with reactive closed-loop predictions without concerns for the time and memory constraints of onboard execution.

5.2.5 Learning from Simulation

Many prior works have attempted to mitigate the issues with IL by training in a simulated environment in order to explore the effects of different counterfactual behaviors for the ego-vehicle. Several of these approaches only roll out the other agents in the scene with log-replay [15] even though the ego-agent is rolled out closed-loop and can take different actions than those in the driving logs. Some approaches attempt to avoid issues with divergence by using some variant of GAIL [46; 4; 9], classmate-forcing [106], or adding an imitation loss to RL [70] in order to encourage the ego-vehicle to visit similar state-actions as those demonstrated in the logs. However, this can be quite limiting as it discourages the agent from gathering information about the effects of performing realistic yet significantly different behaviors from those executed in the driving logs.

On the other hand, there have been a few approaches that have explored using learned models to control the other agents in the scene during offline simulation. Generally these approaches either use behavioral models that were trained to imitate human trajectories from large offline datasets [33; 104] like in our work, or they specifically train adversarial behavior models [43; 34] that are optimized to cause negative outcomes for the ego-vehicle. Our work is most similar to Feng et al. [33], which uses pre-trained traffic forecasting models autoregressively to generate traffic scenarios. However, they then train their reinforcement learning policy from scratch in the generated scenario, while we instead use our pre-trained multimodal forecasting model as a skill model to help guide exploration and provide an implicit behavioral constraint that should help avoid potential model exploitation. Approaches that learn against adversarial agents [43] should theoretically have better worse-case performance, but are inclined to learn overly conservative behaviors in interactive environments, as they will learn to assume that all agents will act adversarially. This could potentially exacerbate the "frozen robot problem" that already occurs in traditional autonomy stacks. Symphony [50] is another approach that is similar to ours in that it uses reactive closed-loop simulation to train a motion planning policy, except that it uses beam-search over the entire joint trajectory to determine good behavior. The potential issue with this approach is that optimizing for a single good future joint trajectory in a stochastic environment like autonomous driving can lead to an over-optimism bias (as we discussed in Chapter 3) in situations where the surrounding vehicles could actually have realistically taken different actions that would have led to much worse outcomes then the one found positive trajectory.

5.3 Background

Autonomous driving is a multi-agent setting where all agents are trying to accomplish their own hidden goals while avoiding negative joint interactions like dead-locks, near misses, or collisions. Thus, autonomous driving can be represented as a general sum Markov game [66; 50] $\mathcal{G} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, \{r^i\}_{i=1}^N, \rho_0\}$. $\mathcal{S}$ is the joint state space; $\mathcal{A}$ is the joint action space; $\mathcal{P}$ is the transition dynamics; $\{r^i\}_{i=1}^N$ are the separate reward functions for each of the $i \leq N$ actors; ρ_0 is the joint initial state distribution. An important caveat is that each agent $i \in N$ only partially observes the scene with observation o_t^i at each time step t and takes a conditionally independent action drawn from its policy distribution $a_t^i \sim \pi^i(\cdot|o_t^i)$. We will denote the kinematic state of each agent like its relative position, orientation, bounding-box, and velocity compared to the ego-vehicle at time 0 as s_t^i. By convention, we will assume that the ego-vehicle will always have index 1 and whose kinematic state is given by s_t^1, while all other agents will have an index $2 \leq i \leq N$.

In this work, we specifically use a hierarchical policy formulation where we explicitly break down our policy into a discrete set of K observation-conditioned skills (low-level policies) and a high-level policy that selects which of the discrete skills to execute in a given state. Thus, our hierarchical policy has the following form

$$\pi^i(a_t^i|o_t^i) = \underbrace{\pi_\theta^i(a_t^i|o_t^i; z_t^i)}_{\text{low-level policy}} * \underbrace{\pi_\phi^i(z_t^i|o_t^i)}_{\text{high-level policy}} \tag{5.1}$$

Once we have pre-trained our discrete set of skills, our ultimate objective is to learn a high-level policy specifically for the ego-vehicle π_ϕ^1 that best recovers the expert policy and maximizes expected reward when deployed in the real system. In order to avoid the potential instabilities that come from training a value function in the loop of a learned world model, we train our policy to optimize the reward over our finite simulation horizon of $H = 8$ steps at 2Hz. Thus, we ultimately train our high-level policy in order to optimize the following objective

$$\max_{\phi} \mathbb{E}\left[\sum_{h=1}^{H} r^1(s_{t+h}, a_{t+h})\right] \tag{5.2}$$

We believe that this is not a limiting assumption, as autonomous driving systems have generally been able to achieve strong performance with limited horizon planning. However, we leave further exploration of learning the value function in the loop as an interesting direction for future work.

We assume access to a reasonably strong perception system that can take in the onboard sensor readings like lidar and images to output estimates of all the surrounding vehicles, bicycles, and pedestrians' kinematic states $\mathcal{X}_t$, if they are within the visibility range of 50m. Additionally, we assume access to both static and dynamic map information $\mathcal{C}_t$ like relatively dense map points, route points, traffic cones, and the location and state of traffic lights.

5.4 Extracting Skills from a Multimodal Forecasting Model

In this section, we will discuss how we can leverage the discrete set of predictions produced by a multimodal traffic forecasting model in order to produce a discrete set of observation-conditioned skills for hierarchical reinforcement learning in autonomous driving.

5.4.1 Low-Level Policies from Multimodal Trajectory Forecasting

In order to train a forecasting model with parameters θ, we assume we have access to a large dataset $\mathcal{D}_\beta$ of logged demonstrations of multi-agent traffic. These demonstrations include state trajectories $\tau^i = \{s^i_0, s^i_1, ..., s^i_T\}$ for each of the agents $i \leq N$ visible to the ego-vehicle while navigating a traffic scene. With this dataset $\mathcal{D}_\beta$ we train our forecasting model to maximize the likelihood of the observed trajectory data for all the vehicles and bicycles in the scene.

$$\max_{\theta} \log P^i_\theta(\tau^i_{\text{gt}} | o^i_t) \tag{5.3}$$

As we have previously discussed, each agent in the environment tends to act stochastically due to internal unknown factors like their intended goals, driving styles, and alertness. Therefore, in any particular state we expect the trajectory distribution for all dynamic agents to be multimodal. Many prior works in traffic forecasting [32; 12; 76; 111; 38] have addressed this by predicting a discrete set of potential trajectories for the different agents in the scene. In this work we use a similar structure to prior works Nayakanti et al. [76]; Girgis et al. [38], and use a Transformer-based encoder-decoder structure where we first pass the entire scene into a Transformer [112] encoder to produce an encoding for the entire scene

$$\mathcal{E}_t^i = \text{Enc}_\theta(\mathcal{X}_t^i, \mathcal{C}_t^i) \tag{5.4}$$

where $(\mathcal{X}_t^i, \mathcal{C}_t^i)$ comes from the observation o_t^i for agent i. Specifically, $\mathcal{X}_t^i$ are the relative states of the dynamic agents in the scene like the vehicles and bicycles, $\mathcal{C}_t^i$ are the features of the map context objects like road and route points, and $\mathcal{E}_t^i$ is the outputted scene encoding from the perspective of agent i. Then, we produce multiple distinct trajectory predictions for each agent by conditioning the decoder on one of a discrete set of K learned anchor embeddings z.

$$\hat{\tau}_z^i = \text{Dec}_\theta(\mathcal{E}_t^i, z) \tag{5.5}$$

Each of these trajectory predictions $\hat{\tau}_z^i$ takes the form of a timeseries of future positions and orientations for that specific agent $\hat{\tau}_z^i = \{\hat{s}_{t+1}^i, \hat{s}_{t+2}^i, ..., \hat{s}_{t+H}^i\}_z$. By decoding with each of the K learned anchor embeddings z, we are able to produce K distinct trajectory predictions. Each of these trajectory predictions is represented by a Gaussian with a predicted mean and variance. Therefore, our models ultimately predict a Gaussian mixture-model (GMM) distribution. However, instead of directly training these models to maximize the likelihood of the data under this induced GMM distribution, we follow prior work [76; 38; 111] and instead adopt a winner-takes-all objective [75] where we split the training loss into separate classification and regression losses.

$$\max_\theta \underbrace{\log\left(P_\theta^i(\tau_{\text{gt}}^i | o_t; z^*)\right)}_{\text{regression loss}} + \underbrace{\log\left(P_\theta^i(z^* | o_t)\right)}_{\text{classification loss}} \tag{5.6}$$

Specifically, given the ground truth trajectory τ_{gt}^i for agent i, we assign the index z^* to the index of the predicted trajectory $\hat{\tau}_z^i$ with lowest $l2$ error. Then, we train just that prediction $\hat{\tau}_{z^*}^i$ with regression to match τ_{gt}^i. Additionally, we train a classifier $P_\theta^i(\cdot|o_t)$ over these K different predictions to maximize the likelihood of index z^* with cross entropy-loss.

Now with each of these H step open-loop predictions $\hat{\tau}_z^i$ produced by our forecasting model, we can infer an action a_t^i for the agent at the current time step. During simulation, we do this by assuming that the vehicles will track the first step of the predicted trajectories exactly, and during online execution for the ego-vehicle we use a controller to track the trajectory. Thus, we can use this model to represent a discrete set of observation-conditioned low-level policies with

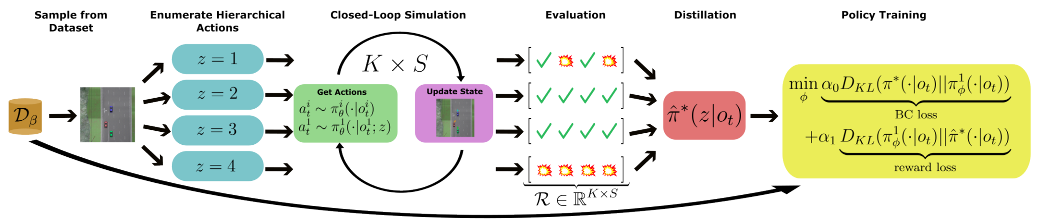
$$\pi^i_\theta(a_t|o_t; z) = \text{Dec}^i_\theta(\text{Enc}_\theta(\mathcal{X}^i_t, \mathcal{C}^i_t), z) \tag{5.7}$$

Additionally, we define the high-level policy $\pi^i_\phi(z^i_t|o^i_t)$ with parameters ϕ as a discrete observation-conditioned policy that selects over the K latents z. Because the potential outputs are discrete, we can simply represent this with a categorical policy. Therefore, we can treat the previously mentioned classifier $P^i_\theta(\cdot|o_t)$ as such a high-level policy, which we will denote as $\pi^i_\theta(\cdot|o_t)$. However, as we have shown in Chapter 4, this high-level policy is not particularly effective at picking which low-level policy to execute for online control. Thus, in Section 5.5.1 we will discuss how we can learn a better high-level policy by evaluating the different observation-conditioned skills with offline simulated rollouts collected in our traffic forecasting world model.

While our HOLOS approach could be used with any multimodal forecasting model that follows the above structure and makes a discrete set of predictions for each agent by conditioning on a discrete set of latent variables, we specifically use our architecture from Chapter 4 in Section 4.3. To adapt the architecture to nuPlan, the main structural change we make is that our model takes in a history of the past states for the dynamic agents, rather than just their current state. To account for this, we add a small 1d CNN that processes these timeseries in order to generate the initial embedding for each agent.

5.4.2 Connections to VQ-VAE

We note that Equation 5.6 is reminiscent of the structure used in many hierarchical RL approaches. In particular, it is similar to prior works that have explored using a VQ-VAE [110] in order to train a discrete set of skills from offline datasets [71; 113]. The difference is that when learning a discrete set of skills offline with a VQ-VAE, one picks which latent prediction to update based on whichever of the latents is closest to the output of the current posterior model. This is similar to our approach in that it introduces a hard cutoff that causes only one of the potential discrete set of predictions to be updated. However, with VQ-VAEs this hard cutoff is done in the latent space, while in our approach this cutoff is determined by which of the discrete predictions is closest in actual prediction space. We hypothesize that our method might be a more effective way of learning skills offline for a



Sample from Dataset
Enumerate Hierarchical Actions
Closed-Loop Simulation
Evaluation
Distillation
Policy Training
$\mathcal{D}_\beta$
$z = 1$
$z = 2$
$z = 3$
$z = 4$
$K \times S$
Get Actions
$a_t^i \sim \pi_\theta^i(\cdot|o_t^i)$
$a_t^1 \sim \pi_\theta^1(\cdot|o_t^1; z)$
Update State
$\mathcal{R} \in \mathbb{R}^{K \times S}$
$\hat{\pi}^*(z|o_t)$
$\min_\phi \underbrace{\alpha_0 D_{KL}(\pi^*(\cdot|o_t)||\pi_\phi^1(\cdot|o_t))}_{\text{BC loss}}$
$+\alpha_1 \underbrace{D_{KL}(\pi_\phi^1(\cdot|o_t)||\hat{\pi}^*(\cdot|o_t))}_{\text{reward loss}}$

Figure 5.2: This figure depicts the steps involved in using our offline simulation to train the hierarchical policy π_ϕ^1 for the ego-vehicle.

hierarchical policy, as it more directly encourages each distinct skill to further specialize in the actions it is already good at predicting. However, we leave further theoretical analysis into these comparisons as an interesting direction for future work. Still, later we show experimentally that our approach to offline skill learning leads to superior online performance in an interactive autonomous driving setting.

5.5 Learning a High-Level Policy from Offline Simulation

In this section, we discuss how we use pre-trained forecasting models in order to perform offline reactive simulation and evaluation in order to learn our high-level policy. For an overview of our HOLOS approach, we include a diagram in Figure 5.2.

5.5.1 Learning Offline to Select Skills

We train the high-level skill selection policy by learning a new discrete categorical policy π_ϕ^1 with parameters ϕ to replace the pre-trained classifier from the forecasting model. As with the pre-trained classifier, this new discrete high-level policy π_ϕ^1 will select over the different K modes of behavior from the ego-forecasting model. In order to avoid potential issues with model exploitation, we just use these offline simulations in order to train our high-level policy and we do not fine-tune the lower-level policies $\pi_\theta^1(a_t|o_t; z)$. Keeping the low-level skills fixed should act as an implicit policy constraint that keeps the overall hierarchical policy within the support of the data. During simulation, we assume the ego-vehicle commits to one of the discrete skills for the entire short simulation horizon H. In addition to providing further regularization, committing to a specific low-level policy for the entire simulation horizon has the added benefit that it simplifies the simulated data collection procedure for training. Still, by having the ego-agent commit to a skill that is different from the one demonstrated in the logged dataset, we can explore the results of different counterfactual actions.

5.5.2 Reactive Simulation

When performing these offline simulations to evaluate counterfactual ego-vehicle behaviors, it is important that the surrounding traffic react to the new actions taken by the ego-vehicle. Thus, in this subsection we will describe how we can use our pre-trained forecasting models in order to perform closed-loop reactive simulation.

First, we initialize the traffic scene based on a specific traffic instance from the logs. Then, we fix the latent mode z for the high-level ego-action. At each step we query just the low-level ego-policy with the current observation of the environment and determined mode z to get its next action $a_t^1 \sim \pi_\theta^1(\cdot|o_t^1; z)$. Additionally at each step, we independently sample both a high-level and corresponding low-level action for each of the other vehicles in the scene $a_t^i \sim \pi_\theta^i(\cdot|o_t^i)$ according to our pre-trained forecasting model. We use a constant velocity assumption for forecasting pedestrians, and we assume that traffic lights will not change from their initial state during simulation. Then, we update the observations for all the agents in the scene and mask any vehicles that leave the visibility range. We repeat this until we reach the end of our relatively short simulation horizon of 4s, which should reduce the potential effects of divergence due to inaccuracies from the learned models.

An important thing to note is that the learned models for both the ego-vehicle and the surrounding traffic are queried at every timestep in simulation, so the marginal trajectory distribution for all agents should react to the current conditions. For example, the likelihood of another vehicle yielding should dramatically increase if it sees another vehicle merging or asserting in front of it. Prior work [104; 92] has demonstrated that this type of closed-loop reactivity should improve the quality and realism of the joint traffic predictions. Specifically for this work, this reactivity should allow the ego-vehicle to explore and better evaluate the effects of non-trivial counterfactual actions in a replayed scene.

5.5.3 Evaluation

We then evaluate these different joint forecastings of the future using a heuristically designed reward function just for the ego-vehicle and its interactions with the different agents. Because the other agents in the scene will react stochastically to the ego-vehicle, we cannot properly evaluate the expected reward for a specific counterfactual ego-behavior with just 1 simulated trajectory in the scene. Therefore, when evaluating a specific high-level action z

for the high-level policy, we simulate $S > 1$ different stochastic closed-loop trials. In each of these S trials, we sample independently from the marginal distribution for each agent in the scene at each timestep and thus should get S different scene outcomes drawn from the joint traffic distribution given that the ego-vehicle is executing the specific skill corresponding to the low-level policy $\pi^1_\theta(\cdot|o_t; z)$. Thus to evaluate the high-level policy's K different high-level actions for an initial scene, we run $K \times S$ different closed-loop simulations and coalesce the resulting rewards into a matrix $\mathcal{R} \in \mathbb{R}^{K \times S}$ that we can use for policy training.

5.5.4 Policy Training

Now for training the high-level policy we take inspiration from maximum-entropy RL [53; 128] and assume that an expert would take actions proportional to the exponential of their expected reward.

$$\pi^*(z|o_t) \propto \exp \mathbb{E}\left[\sum_{h=0}^{H} r(s_{t+h}, a_{t+h})\right] \tag{5.8}$$

We can estimate the expected reward for each possible high-level action for the high-level policy by taking the mean over the S different samples collected in simulation. Thus, we can represent the expert policy by taking a softmax over the mean rewards for each mode.

$$\hat{\pi}^*(z|o_t) = \text{softmax}\left(\frac{1}{S}\sum_{s=1}^{S} \mathcal{R}_{zs}\right) \tag{5.9}$$

Here $z \leq K$ corresponds to the ego-mode, and $s \leq S$ corresponds to a specific simulated trial.

Following SAC [41], we train our policy to minimize the reverse KL divergence with $\hat{\pi}^*$. Because we are using discrete distributions and have an analytic form for $\hat{\pi}^*$, we can compute this loss exactly.

$$\min_\phi D_{KL}(\pi^1_\phi(\cdot|o_t)||\hat{\pi}^*(\cdot|o_t)) \tag{5.10}$$

Note that this target policy $\hat{\pi}^*(z|o_t)$, is almost equivalent to a stochastically smoothed version of running our sample-based planner from Chapter 4 (P2DBM) on the offline sample corresponding to s_t. Thus, the loss in Equation 5.10 is analogous to training our high-level policy to distill the results of our P2DBM planner ran on the offline samples in the training set.

In settings where we assume the initial trajectories from $\mathcal{D}_\beta$ come from an actual expert policy π^* we can include an additional behavioral cloning (BC) loss to fit z^*, which corresponds to the mode closest to the ground-truth trajectory. This loss is equivalent to fitting the forward KL-divergence with π^*, but approximated with the limited samples from the logged dataset $\mathcal{D}_\beta$. Thus, our final loss for training our high-level policy is

$$\begin{aligned} \min_{\phi} \underbrace{D_{KL}(\pi^1_\phi(\cdot|o_t)||\hat{\pi}^*(\cdot|o_t))}_{\text{reward loss}} \\ +\alpha \underbrace{D_{KL}(\pi^*(\cdot|o_t)||\pi^1_\phi(\cdot|o_t))}_{\text{BC loss}} \end{aligned} \tag{5.11}$$

where α corresponds to the weights for the optional BC loss.

Prior work in Rhinehart et al. [89] illustrates the benefits and effectiveness of using a symmetric divergence for training, like our loss function in equation 5.12. Training with each of the two directions of the KL-divergence has different yet complementary biases on fitting the underlying distribution. The forward direction, often used in behavioral cloning, encourages ”mode covering” even if this leads to an overly diffuse distribution that places high probability mass on samples outside of the support of the data. On the other hand, the reverse direction, often used in RL, encourages ”mode seeking” and greatly penalizes these out-of-distribution samples. In motion-planning for autonomous driving, training with this symmetric divergence should correspond to the high-level policy still covering the modes of reasonable behavior while putting near zero probability on non-safe actions that could lead to collisions or other serious infractions.

In order to further facilitate training the high-level policy, the high-level policy shares the pre-trained encoder from the forecasting model and focuses on training just a new Transformer-based decoder that outputs logits for each of the K different skills. We represent the categorical policy with a standard softmax distribution and train the model with the loss from equation 5.12. During online execution, in order to maximize reactivity, we requery the maximum-likelihood action from the high-level policy at every step and execute the action from the corresponding low-level policy.

$$\hat{z}^1_t = \max_{z^1_t} \log(\pi^1_\phi(z^1_t|o^1_t)) \tag{5.12}$$

$$\hat{a^1_t} = \max_{a^1_t} \pi^1_\theta(a^1_t|o^1_t;\hat{z}^1_t) \tag{5.13}$$

5.6 Experiments

5.6.1 CARLA Merge Scenarios

We first validate our approach in the same curated set of hand-designed merging scenarios run in version 0.9.11 of the CARLA [31] simulator, as in Chapter 4. These scenarios require a high degree of interactivity with multiple other agents in the traffic scene in order to succeed. The other agents in the scene react stochastically, as they are controlled by their own adaptive cruise controllers with randomized parameters, roughly corresponding to different levels of aggressiveness. In many of these scenarios, the ego-vehicle must assert itself, knowing that the other vehicles will react and slow down for it given enough space. However, depending on the traffic conditions and the stochasticity of the other agents, the ego-vehicle needs to be able to reason about when it is truly safe to assert itself, and when it would be better to yield.

In this setting, the neural networks take in the positions, orientations, bounding boxes, and velocities of all the visible agents in the scene and do not track their histories. These scenarios just focus on vehicle interactions, and thus do not include any pedestrians, stop lights, or stop signs. The map information $\mathcal{C}_t$ includes dense points following the lane centers and dense route points for the ego-vehicle to follow. The pre-trained forecasting models output $K = 8$ discrete modes, and we use $S = 8$ samples to evaluate expectations during offline simulations. We collect potentially sub-optimal demonstrations with a stochastic version of the rule-based autopilot from Prakash et al. [83] with the target speed set to the current speed limit. Because we do not assume the trajectories come from an optimal policy, we use $\alpha = 0$ for the loss function in updating the high-level policy in equation 5.12. For policy training and planning, we use the same reward function as in Chapter 2. We collect a dataset of 100k samples in order to pre-train our forecasting model and to use as initial states for our offline simulations. In order to avoid trivial solutions where the vehicle waits for all traffic to pass before attempting to merge, we include static as a failure case when the car freezes for several seconds before the merge point. We report results of performing 20 different trials in 10 different merging scenarios run over 3 different model seeds.

We compare to a variety of different baselines in order to validate the effectiveness of our method for offline skill learning, and our Hierarchical Offline Learning with Offline reactive Simulation (HOLOS) method for learning our high-level policy:

- *Replay Sim* uses the same training procedure as our approach, except it simulates the other vehicles using log-replay.

Approach	Success (%) ↑	Static (%) ↓	Crash (%) ↓	Speed (m/s) ↑	Inf Time (ms) ↓
Ours	96.3 ± 1.9	2.2 ± 1.4	1.5 ± 0.5	19.6 ± 0.0	14.6 ± 0.0
Replay Sim	94.5 ± 1.3	**1.0** ± **0.3**	4.5 ± 1.6	19.8 ± 0.1	14.6 ± 0.0
NR Sim	89.8 ± 0.6	8.8 ± 0.3	1.3 ± 0.4	18.3 ± 0.1	14.5 ± 0.0
IL	53.3 ± 1.4	43.8 ± 0.3	2.8 ± 1.6	14.9 ± 0.1	**13.0** ± **0.0**
VQ-VAE	89.2 ± 4.2	7.3 ± 3.8	3.5 ± 1.3	**20.4** ± **0.2**	15.6 ± 0.0
IQL	88.8 ± 2.2	4.3 ± 0.9	6.8 ± 2.5	18.7 ± 0.2	17.9 ± 0.0
P2DBM	**97.7** ± **0.8**	1.2 ± 0.4	**1.2** ± **0.4**	20.0 ± 0.1	196.3 ± 0.3
Data Policy	72.5	11.0	16.5	18.6	16.2

Table 5.1: We report results with averages and standard errors over 3 different seeds of running the 200 different episodes. ↑ and ↓ indicates that higher or lower numbers are better respectively. We bold the results with the best mean.

- *Non-reactive (NR) Sim* uses the same training procedure as our approach, except it simulates the other vehicles using sampled open-loop trajectories predicted from the first time step. Thus, the other vehicles will display a variety of counterfactual behaviors, but will not react to the ego-vehicle's actions.

- *Imitation Learning (IL)* executes the maximum likelihood action according to the pre-trained forecasting model.

- *VQ-VAE* uses the same high-level policy training procedure as our approach, except it use a VQ-VAE [110] model for learning the discrete skills for both the ego-vehicle and surrounding agents.

- *Implicit Q Learning (IQL)* involves training a Q function using the offline RL algorithm IQL [58]. Then, it executes the proposed action from the pre-trained forecasting model with the highest Q value.

- *P2DBM* is our online planning approach from Chapter 4. In this work, we make a minor modification where we resample the latent modes for each surrounding agent at every time step, instead of sampling the latent modes for those agents at just the first time step. Our HOLOS approach is analogous to distilling this version of our P2DBM planner run on our offline dataset. Therefore, P2DBM represents an upper-bound on our performance.

- *Data Policy* is the stochastic rule-based policy used to collect the dataset.

We report our results in Table 5.1 and find that all the approaches that learn the high-level policy from offline simulations significantly outperform doing just standard IL. However, our approach of offline high-level policy learning using stochastic and reactive simulation does best in reducing the gap with online planning. We see that learning from non-reactive simulation leads to overly cautious behavior with a higher static rate and lower average speed because it fails to anticipate when other vehicles will yield for it. On the other hand, learning from simulation with log-replay for the other agents leads to overly aggressive behavior with a higher crash rate and higher average speed, presumably because the ego-agent overfit to the specific logged trajectories during training and does not anticipate the stochastic behavior of the other agents. Additionally, we find that running our high-level policy training algorithm with the discrete set of skills from our offline skill learning approach outperforms running our algorithm with skills learned from a VQ-VAE. Finally, we outperform our representative offline RL baseline of IQL.

While using the forecasting model for online planning does lead to the best performance, it would be impractical to use on a real system due to its prohibitive inference time of 196 milliseconds, shown in the rightmost column of Table 5.1. During tight interactive scenarios or when traveling at high speed, we would require a much higher control frequency in order properly react to the evolving scene. Therefore, our approach, which just needs one pass of the hierarchical policy to produce an action and is over **10x** faster than the planner, would be more suitable for online deployment.

5.6.2 nuPlan

In order to validate the scalability of our approach, we also report results on the nuPlan benchmark [40]. This is a large-scale benchmark that aims to evaluate the closed-loop performance of machine learning-based motion planners. The dataset consists of 1200 hours of human driving data collected from 4 cities in the US and Asia. Additionally, nuPlan provides an open-source simulator for simulating various real traffic scenes in a closed-loop manner. We evaluate results in the reactive closed-loop setting, where the other agents use a rule-based policy to dynamically react to the ego-vehicle and other agents. We follow Cheng et al. [17] and report results on two different splits of the public test set: Test14-random and Test14-hard.

For our nuPlan results, the neural networks take in the positions, orientations, and bounding boxes of all the visible agents in the scene and do track the last 2s of history at 2Hz. We only train the forecasting model to make predictions for vehicles and bicyclists, but we also include pedestrians and traffic cones as dynamic agents in the scene. For map context $\mathcal{C}_t$, we include dense map points along the road centers and route points. All the

Approach		Test14	
Type	Method	random ↑	hard ↑
Expert	Log-replay	75.86	68.80
Rule-based	IDM†	72.42	62.26
	PDM-Closed†	**91.64**	75.18
Hybrid	GameFormer†	79.31	68.83
	PDM-Hybrid†	91.56	**75.79**
	P2DBM (Ours)*	84.05	67.97
Learning	RasterModel†	67.54	52.16
	UrbanDriver†	61.02	49.07
	PDM-Open†	57.23	35.83
	PlanTF†	80.59	61.70
	IL (Ours)*	71.65	53.51
	Replay Sim (Ours)*	75.22	60.83
	HOLOS (Ours)	81.58	63.54

Table 5.2: We report nuPlan driving scores on the Test14-random and Test14-hard splits, run with reactive closed-loop simulation (R-CLS). Higher is better, with a max theoretical score of 100. For each of the Test14-random and Test14-hard splits, we bold the best results. The approaches denoted with * are our own implemented baselines that leverage the same pre-trained forecasting model as our approach, but in different ways. For approaches denoted with †, we use the reported results from the PlanTF paper.

map points include relative positions and orientations to the next map point, and the road center points include additional information about whether they are affected by a traffic light. The pre-trained forecasting models outputs $K = 16$ discrete modes, and we use $S = 8$ samples to evaluate expectations during offline simulation. Because we assume the trajectories come from expert demonstrations, we use $\alpha = 1$ for the loss function in updating the high-level policy in equation 5.12. For both pre-training the forecasting model, and training the high-level policy we use 1% of the training data, which corresponds to roughly $260K$ samples.

In Table 5.6.2, we compare to a variety of different rule-based [109; 29], hybrid learning with planning [49; 29], and strictly learning-based approaches [40; 94; 29; 17] and include the results reported in [17]. The most relevant approach for our comparisons is PlanTF, which is a well-tuned and ablated IL-based approach that uses a Transformer [112] backbone. To the best of our knowledge, PlanTF is the state of the art on solely learning-based motion planning for closed-loop simulation in nuPlan.

Similar to our CARLA results, we find that there is still a gap between our learning-based approach's performance and the performance of approaches that incorporate online planning. However, we find that our approach is the closest to bridging that gap and outperforms all the other learning-based approaches on the reactive closed-loop evaluations. We again outperform *Replay Sim*, which uses the same training procedure as our approach, except it simulates the other vehicles using log-replay. We even outperform PlanTF, which uses a similar Transformer-based backbone to ours, but was specifically designed and ablated to get the best IL-based results.

5.7 Conclusion

In this chapter, we presented our Hierarchical Offline Learning with Offline reactive Simulation (HOLOS) approach. We illustrated how we could extract a discrete set of low-level skill policies from multimodal traffic forecasting models that were pre-trained on offline datasets. Additionally, we demonstrated how we could train a high-level policy that learns to selects over this discrete set of skills with offline reactive simulation. In particular, we used our pre-trained multimodal forecasting model as a world model to predict different potential stochastic responses from the other dynamic agents in the scene in closed-loop. Then, we distilled the results of running these many stochastic and reactive simulations for all of the discrete skills into our high-level policy. Ultimately, we demonstrated how our approach was the closest to reducing the gap between a purely learning-based motion planning policy and hybrid approaches that include online planning. Additionally, we demonstrate how our approach can outperform a state-of-the-art IL approach in the extensive nuPlan benchmark.

CHAPTER 6

Conclusion

In this thesis, we explored how we can learn deep models offline from previously collected datasets of driving trajectories in order to facilitate online motion planning in autonomous driving. In particular, we explored different ways in which we could train and use these models in order to better handle stochastic interactions with unknown agents. We first demonstrated how we can use offline model-based RL to collect additional counterfactual rollouts in order to improve an offline-learned policy in a deterministic robotic locomotion setting. Next, we transitioned to a stochastic autonomous driving setting and showed how we can avoid optimism bias by appropriately disentangling the effects of the policy and the world model on the resulting reward during online planning. We found, however, that our simple feature-based observation space was not sufficient to handle more complicated interactions in autonomous driving. Thus, we moved on to using a multi-agent formulation of the environment with a Transformer-based multimodal traffic forecasting model acting as the world model. With this multi-agent formulation, we demonstrated how we could efficiently do mutually reactive and stochastic online planning by explicitly planning over the different discrete behavior predictions from the multimodal forecasting model. Finally, we presented an offline hierarchical learning algorithm that learned which high-level behavior to execute by training with stochastic closed-loop rollouts collected in our traffic forecasting world model.

We believe the work in this thesis and other ongoing recent works in the field contribute to improving the capability for learning-based motion planners to engage in proactive yet safe behaviors in stochastic interactive scenarios. However, there are still several areas that need further exploration. Below, we discuss some challenges we faced during our work, as well as potential future directions for research.

6.1 Learning for Motion Planning

While we found that our offline hierarchical approach was able to outperform other purely machine learning-based motion planning policies, our approach still underperformed by a large margin when compared to a more traditional heuristic-based planning approach. In fact, the PDM [29] planner approaches actually won the nuPlan competition and beat out a variety of hybrid approaches that use learning to guide online planning. Both our results and these results indicate that more future work needs to be done in order to improve the capabilities of machine learning-based motion planners for autonomous driving.

Specifically, we found that many of the failure cases for our P2DBM approach in nuPlan came from situations in which the underlying forecasting model failed to predict a drastic action needed to resolve the current situation. For example, the model would fail to predict an aggressive lane change or sudden brake in order to avoid a collision with a vehicle that unexpectedly cut in front of the ego-vehicle. We suspect that this issue stems from some version of the "copy-cat problem" [117] or causal confusion [30], which is exacerbated by the fact that these drastic actions are not well represented in the data. Therefore, it is difficult for these models, trained to maximize the likelihood of the data, to reliably output these more drastic actions, even though these actions are extremely necessary to handle the long-tail of potentially dangerous scenarios.

One potential direction to alleviate these issues would be to try to rebalance the dataset in order to increase the prevalence of these less common drastic actions during training. This should greatly increase the likelihood that the learned models produce these important actions during critical states. Another interesting potential direction would be to use a more expressive distribution for the ego-vehicle in combination with a more intelligent sampling procedure, like non-maximum suppression [111]. This would allow us to explicitly sample more diverse predictions from the learned model and improve the coverage of the proposed candidates. Seff et al. [98] is one recent work that has explored these ideas in the context of autoregressive traffic forecasting, but they do not explore how this could be used to facilitate online motion planning. We leave how to do this efficiently and effectively in a mutually reactive planning approach as a potential avenue for future exploration.

6.2 Evaluating with Simulation

One of the limitations of being an independent academic researcher is that we had no feasible means of evaluating our approaches in real-world driving situations. While we were able to gain valuable insights by testing in closed-loop simulated environments like CARLA [31] and nuPlan [40], there are definitely large gaps between these simulated environments and the real world. In particular, the controllers used for the other vehicles in the environment often behaved unnaturally in these more interesting interactive scenarios that we focused on in this thesis. For example, the traffic-managed vehicles in CARLA would not react to the ego-vehicle until the ego-vehicle entered the same lane. Afterwards, however, the traffic-managed vehicle would often overreact and abruptly brake even when there was plenty of headway for the ego-vehicle. Initially, we hoped that these unnatural behaviors would be ameliorated in the nuPlan benchmark because it is based on real-world traffic trajectories. While the traffic scenarios in nuPlan were significantly more varied and realistic, the IDM controllers used to control the other vehicles in the closed-loop reactive simulations still exhibited similar issues to those in CARLA. Unfortunately, generating realistic and responsive traffic for evaluating an autonomous system seems to be a "chicken and egg" problem, since developing realistic and responsive behaviors for the other vehicles is equally as difficult as producing good behaviors for the ego-vehicle. Therefore, I am excited to see how advances in work like Montali et al. [73], Igl et al. [50], Bhattacharyya et al. [7], and Xu et al. [123] that look at learning models for performing more accurate closed-loop traffic simulation can be used to better evaluate autonomous systems and for training RL agents in the loop.

6.3 Computational Efficiency

I think that joint prediction and planning will be critical in enabling autonomous driving systems to smoothly and naturally interact with human drivers on the road. While there are inherent computational difficulties in searching over the entire space of joint interactions between all of the different agents, I am optimistic that machine learning can be used to help guide and prune the search process. In this thesis, we took some initial steps in exploring how learning multimodal forecasting models from logged trajectories can be used to efficiently reduce the search space. However, we believe that there is a lot of work to be done in order to further improve the efficiency and efficacy of using these models in planning.

Specifically, we want to reduce the runtime of these systems as much as possible in order to maximize their responsiveness. While autoregressive Transformer models can achieve impressive results in multi-agent motion forecasting [104; 98], the compute needed to use these large models scales poorly with respect to the number of agents and time steps. We believe an important direction for future work will be to explore different ways in which we can get the benefits of these types of models, while improving their computational efficiency. Some promising approaches include using more efficient architectures like Varadarajan et al. [111], Nayakanti et al. [76], or some of the various efficient Transformer architectures that try to circumvent the quadratic scaling of attention [107]. Additionally, autoregressive architectures that avoid running the entire neural network model for each future prediction could be a promising direction for reducing the computational scaling in time [98].

Another direction would be to use approaches that attempt to directly perform joint predictions with one pass of the neural network model [77; 104]. While using these models would reduce the computational complexity, such approaches will have to be careful to avoid the optimism bias that we discussed in this thesis. For example, joint predictions for the ego-vehicle and surrounding traffic will be biased to predict that the surrounding traffic will engage in behaviors that are harmonious with the chosen ego-vehicle behavior because most traffic datasets only contain positive scene outcomes. But if these potential biases can be addressed, directly planning over joint predictions could be an efficient means of using learning to guide motion planning.

6.4 On-Policy Samples

The success of ChatGPT [1], Alpha-Zero [101], and many works in fine-tuning offline-learned models with online RL [70; 58; 30] illustrate the importance of collecting on-policy samples in order to continually improve models and reduce failures. While we believe that it would be dangerous and impractical to learn a RL driving policy from scratch in the real-world, we do think that it will be important to leverage on-policy samples in order to continually improve driving performance on the long-tail of scenarios. In this thesis, we did some initial work in using on-policy samples collected offline in a simulated world model in order to improve a hierarchical policy. However, there is inevitably no replacement for real-world driving data. Therefore, we believe that fully utilizing and learning from trajectories with interventions and other on-policy failures from actual autonomous systems run in the real-world will be invaluable to improving their performance.

6.5 Final Remarks

In this thesis, we presented our work on how we can use deep learning models to facilitate safe yet proactive interactions with other stochastic vehicles. While we believe there are still many challenges and open questions, I am optimistic that with the ingenuity of the wider research community, we can resolve many of these issues I raised, and I am excited for the future of deep learning-based motion planners for autonomous driving.

BIBLIOGRAPHY

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] A. Ajay, A. Kumar, P. Agrawal, S. Levine, and O. Nachum. Opal: Offline primitive discovery for accelerating offline reinforcement learning. *ICLR*, 2021.

[3] M. Bansal, A. Krizhevsky, and A. Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. In *RSS*, 2019.

[4] N. Baram, O. Anschel, I. Caspi, and S. Mannor. End-to-end differentiable adversarial imitation learning. In *International Conference on Machine Learning*, pages 390–399. PMLR, 2017.

[5] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012. URL http://arxiv.org/abs/1207.4708.

[6] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[7] R. P. Bhattacharyya, D. J. Phillips, B. Wulfe, J. Morton, A. Kuefler, and M. J. Kochenderfer. Multi-agent imitation learning for driving simulation. in 2018 ieee. In *RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1534–1539.

[8] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[9] E. Bronstein, M. Palatucci, D. Notz, B. White, A. Kuefler, Y. Lu, S. Paul, P. Nikdel, P. Mougin, H. Chen, et al. Hierarchical model-based imitation learning for planning in autonomous driving. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8652–8659. IEEE, 2022.

[10] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[11] S. Casas, C. Gulino, S. Suo, K. Luo, R. Liao, and R. Urtasun. Implicit latent variable model for scene-consistent motion forecasting. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIII 16*, pages 624–641. Springer, 2020.

[12] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, et al. Argoverse: 3d tracking and forecasting with rich maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8748–8757, 2019.

[13] D. Chen and P. Krähenbühl. Learning from all vehicles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17222–17231, 2022.

[14] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020.

[15] D. Chen, V. Koltun, and P. Krähenbühl. Learning to drive from a world on rails. *arXiv preprint arXiv:2105.00636*, 2021.

[16] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *arXiv preprint arXiv:2106.01345*, 2021.

[17] J. Cheng, Y. Chen, X. Mei, B. Yang, B. Li, and M. Liu. Rethinking imitation-based planner for autonomous driving. *arXiv preprint arXiv:2309.10443*, 2023.

[18] K. Chitta, A. Prakash, and A. Geiger. Neat: Neural attention fields for end-to-end autonomous driving. In *International Conference on Computer Vision (ICCV)*, 2021.

[19] S. Choi, J. Kim, and H. Yeo. Attention-based recurrent neural network for urban vehicle trajectory prediction. *Procedia Computer Science*, 151:327–334, 2019.

[20] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *CoRR*, abs/1805.12114, 2018. URL http://arxiv.org/abs/1805.12114.

[21] F. Codevilla, E. Santana, A. M. López, and A. Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9329–9338, 2019.

[22] F. Codevilla, E. Santana, A. M. López, and A. Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9329–9338, 2019.

[23] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[24] A. Cui, S. Casas, A. Sadat, R. Liao, and R. Urtasun. Lookout: Diverse multi-future prediction and planning for self-driving. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 16087–16096. IEEE Computer Society, 2021.

[25] A. Cui, S. Casas, K. Wong, S. Suo, and R. Urtasun. Gorela: Go relative for viewpoint-invariant motion forecasting. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7801–7807. IEEE, 2023.

[26] W. Dabney, G. Ostrovski, D. Silver, and R. Munos. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, pages 1096–1105. PMLR, 2018.

[27] R. Dadashi, L. Hussenot, D. Vincent, S. Girgin, A. Raichuk, M. Geist, and O. Pietquin. Continuous control with action quantization from demonstrations. *arXiv preprint arXiv:2110.10149*, 2021.

[28] M. Dalal, D. Pathak, and R. R. Salakhutdinov. Accelerating robotic reinforcement learning via parameterized action primitives. *Advances in Neural Information Processing Systems*, 34:21847–21859, 2021.

[29] D. Dauner, M. Hallgarten, A. Geiger, and K. Chitta. Parting with misconceptions about learning-based vehicle motion planning. In *Conference on Robot Learning (CoRL)*, 2023.

[30] P. De Haan, D. Jayaraman, and S. Levine. Causal confusion in imitation learning. *Advances in neural information processing systems*, 32, 2019.

[31] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[32] S. Ettinger, S. Cheng, B. Caine, C. Liu, H. Zhao, S. Pradhan, Y. Chai, B. Sapp, C. R. Qi, Y. Zhou, Z. Yang, A. Chouard, P. Sun, J. Ngiam, V. Vasudevan, A. McCauley, J. Shlens, and D. Anguelov. 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)*, pages 9710–9719, October 2021.

[33] L. Feng, Q. Li, Z. Peng, S. Tan, and B. Zhou. Trafficgen: Learning to generate diverse and realistic traffic scenarios. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3567–3575. IEEE, 2023.

[34] S. Feng, H. Sun, X. Yan, H. Zhu, Z. Zou, S. Shen, and H. X. Liu. Dense reinforcement learning for safety validation of autonomous vehicles. *Nature*, 615(7953): 620–627, 2023.

[35] B. Freed, S. Venkatraman, G. A. Sartoretti, J. Schneider, and H. Choset. Learning temporally abstractworld models without online experimentation. In *International Conference on Machine Learning*, pages 10338–10356. PMLR, 2023.

[36] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

[37] M. Ghavamzadeh, M. Petrik, and Y. Chow. Safe policy improvement by minimizing robust baseline regret. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29, pages 2298–2306. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/9a3d458322d70046f63dfd8b0153ece4-Paper.pdf.

[38] R. Girgis, F. Golemo, F. Codevilla, M. Weiss, J. A. D'Souza, S. E. Kahou, F. Heide, and C. Pal. Latent variable sequential set transformers for joint multi-agent motion prediction. *ICLR*, 2022.

[39] V. Gómez, H. J. Kappen, J. Peters, and G. Neumann. Policy search for path integral control. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 482–497. Springer, 2014.

[40] K. T. e. a. H. Caesar, J. Kabzan. Nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles. In *CVPR ADP3 workshop*, 2021.

[41] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL http://arxiv.org/abs/1801.01290.

[42] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[43] N. Hanselmann, K. Renz, K. Chitta, A. Bhattacharyya, and A. Geiger. King: Generating safety-critical driving scenarios for robust imitation via kinematics gradients. In *European Conference on Computer Vision*, pages 335–352. Springer, 2022.

[44] J. Hardy and M. Campbell. Contingency planning over probabilistic obstacle predictions for autonomous road vehicles. *IEEE Transactions on Robotics*, 29(4):913–929, 2013.

[45] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

[46] J. Ho and S. Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

[47] Y. Hu, K. Li, P. Liang, J. Qian, Z. Yang, H. Zhang, W. Shao, Z. Ding, W. Xu, and Q. Liu. Imitation with spatial-temporal heatmap: 2nd place solution for nuplan challenge. *arXiv preprint arXiv:2306.15700*, 2023.

[48] Y. Hu, J. Yang, L. Chen, K. Li, C. Sima, X. Zhu, S. Chai, S. Du, T. Lin, W. Wang, et al. Planning-oriented autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17853–17862, 2023.

[49] Z. Huang, H. Liu, and C. Lv. Gameformer: Game-theoretic modeling and learning of transformer-based interactive prediction and planning for autonomous driving. *arXiv preprint arXiv:2303.05760*, 2023.

[50] M. Igl, D. Kim, A. Kuefler, P. Mougin, P. Shah, K. Shiarlis, D. Anguelov, M. Palatucci, B. White, and S. Whiteson. Symphony: Learning realistic and diverse agents for autonomous driving simulation. *arXiv preprint arXiv:2205.03195*, 2022.

[51] M. Janner, J. Fu, M. Zhang, and S. Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pages 12519–12530, 2019.

[52] M. Janner, Q. Li, and S. Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in Neural Information Processing Systems*, 34, 2021.

[53] E. T. Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4): 620, 1957.

[54] A. Kamenev, L. Wang, O. B. Bohan, I. Kulkarni, B. Kartal, A. Molchanov, S. Birchfield, D. Nistér, and N. Smolyanskiy. Predictionnet: Real-time joint probabilistic traffic prediction for planning, control, and simulation. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8936–8942. IEEE, 2022.

[55] R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims. Morel : Model-based offline reinforcement learning, 2020.

[56] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[57] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[58] I. Kostrikov, A. Nair, and S. Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

[59] S. Krishnan, R. Fox, I. Stoica, and K. Goldberg. Ddco: Discovery of deep continuous options for robot learning from demonstrations. In *Conference on robot learning*, pages 418–437. PMLR, 2017.

[60] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[61] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11784–11794, 2019.

[62] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning, 2020.

[63] R. Laroche, P. Trichelair, and R. T. D. Combes. Safe policy improvement with baseline bootstrapping. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3652–3661, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/laroche19a.html.

[64] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020.

[65] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[66] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

[67] J. Liu, W. Zeng, R. Urtasun, and E. Yumer. Deep structured reactive planning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4897–4904. IEEE, 2021.

[68] Y. Liu, J. Zhang, L. Fang, Q. Jiang, and B. Zhou. Multimodal motion prediction with stacked transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7577–7586, 2021.

[69] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[70] Y. Lu, J. Fu, G. Tucker, X. Pan, E. Bronstein, R. Roelofs, B. Sapp, B. White, A. Faust, S. Whiteson, et al. Imitation is not enough: Robustifying imitation with

reinforcement learning for challenging driving scenarios. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7553–7560. IEEE, 2023.

[71] J. Luo, P. Dong, J. Wu, A. Kumar, X. Geng, and S. Levine. Action-quantized offline reinforcement learning for robotic skill learning. In *Conference on Robot Learning*, pages 1348–1361. PMLR, 2023.

[72] Y. Ma, D. Jayaraman, and O. Bastani. Conservative offline distributional reinforcement learning. *Advances in Neural Information Processing Systems*, 34:19235–19247, 2021.

[73] N. Montali, J. Lambert, P. Mougin, A. Kuefler, N. Rhinehart, M. Li, C. Gulino, T. Emrich, Z. Yang, S. Whiteson, et al. The waymo open sim agents challenge. *Advances in Neural Information Processing Systems*, 36, 2024.

[74] A. Nair, M. Dalal, A. Gupta, and S. Levine. Accelerating online reinforcement learning with offline datasets, 2020.

[75] S. Narayanan, R. Moslemi, F. Pittaluga, B. Liu, and M. Chandraker. Divide-and-conquer for lane-aware diverse trajectory prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15799–15808, 2021.

[76] N. Nayakanti, R. Al-Rfou, A. Zhou, K. Goel, K. S. Refaat, and B. Sapp. Wayformer: Motion forecasting via simple & efficient attention networks. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2980–2987. IEEE, 2023.

[77] J. Ngiam, B. Caine, V. Vasudevan, Z. Zhang, H.-T. L. Chiang, J. Ling, R. Roelofs, A. Bewley, C. Liu, A. Venugopal, et al. Scene transformer: A unified architecture for predicting multiple agent trajectories. *arXiv preprint arXiv:2106.08417*, 2021.

[78] S. Ozair, Y. Li, A. Razavi, I. Antonoglou, A. v. d. Oord, and O. Vinyals. Vector quantized models for planning. *arXiv preprint arXiv:2106.04615*, 2021.

[79] X. B. Peng, A. Kumar, G. Zhang, and S. Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[80] K. Pertsch, Y. Lee, and J. Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on robot learning*, pages 188–204. PMLR, 2021.

[81] J. Peters and S. Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th international conference on Machine learning*, pages 745–750, 2007.

[82] D. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D. Touretzky, editor, *Proceedings of (NeurIPS) Neural Information Processing Systems*, pages 305 – 313. Morgan Kaufmann, December 1989.

[83] A. Prakash, K. Chitta, and A. Geiger. Multi-modal fusion transformer for end-to-end autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7077–7087, 2021.

[84] A. Quintanar, D. Fernández-Llorca, I. Parra, R. Izquierdo, and M. Sotelo. Predicting vehicles trajectories in urban scenarios with transformer networks and augmented information. *arXiv preprint arXiv:2106.00559*, 2021.

[85] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. 2018.

[86] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[87] K. Renz, K. Chitta, O.-B. Mercea, A. S. Koepke, Z. Akata, and A. Geiger. Plant: Explainable planning transformers via object-level representations. In *Conference on Robot Learning*, pages 459–470. PMLR, 2023.

[88] D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

[89] N. Rhinehart, K. M. Kitani, and P. Vernaza. R2p2: A reparameterized pushforward policy for diverse, precise generative path forecasting. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 772–788, 2018.

[90] N. Rhinehart, R. McAllister, and S. Levine. Deep imitative models for flexible inference, planning, and control. *arXiv preprint arXiv:1810.06544*, 2018.

[91] N. Rhinehart, R. McAllister, K. Kitani, and S. Levine. Precog: Prediction conditioned on goals in visual multi-agent settings. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2821–2830, 2019.

[92] N. Rhinehart, J. He, C. Packer, M. A. Wright, R. McAllister, J. E. Gonzalez, and S. Levine. Contingencies from observations: Tractable contingency planning with learned behavior models. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13663–13669. IEEE, 2021.

[93] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[94] O. Scheel, L. Bergamini, M. Wolczyk, B. Osiński, and P. Ondruska. Urban driver: Learning to drive from real-world demonstrations using policy gradients. In A. Faust, D. Hsu, and G. Neumann, editors, *Proceedings of the 5th Conference on Robot*

Learning, volume 164 of *Proceedings of Machine Learning Research*, pages 718–728. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/scheel22a.html.

[95] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver. Mastering atari, go, chess and shogi by planning with a learned model, 2020.

[96] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL http://arxiv.org/abs/1502.05477.

[97] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

[98] A. Seff, B. Cera, D. Chen, M. Ng, A. Zhou, N. Nayakanti, K. S. Refaat, R. Al-Rfou, and B. Sapp. Motionlm: Multi-agent motion forecasting as language modeling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8579–8590, 2023.

[99] T. Shankar and A. Gupta. Learning robot skills with temporal variational inference. In *International Conference on Machine Learning*, pages 8624–8633. PMLR, 2020.

[100] Y. Shen, M. J. Tobia, T. Sommer, and K. Obermayer. Risk-sensitive reinforcement learning. *Neural computation*, 26(7):1298–1328, 2014.

[101] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[102] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015.

[103] H. Song, W. Ding, Y. Chen, S. Shen, M. Y. Wang, and Q. Chen. Pip: Planning-informed trajectory prediction for autonomous driving. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXI 16*, pages 598–614. Springer, 2020.

[104] S. Suo, S. Regalado, S. Casas, and R. Urtasun. Trafficsim: Learning to simulate realistic multi-agent behaviors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10400–10409, 2021.

[105] R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

[106] C. Tang and R. R. Salakhutdinov. Multiple futures prediction. *Advances in Neural Information Processing Systems*, 32:15424–15434, 2019.

[107] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28, 2022.

[108] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[109] M. Treiber, A. Hennecke, and D. Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 62(2):1805, 2000.

[110] A. Van Den Oord, O. Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[111] B. Varadarajan, A. Hefny, A. Srivastava, K. S. Refaat, N. Nayakanti, A. Cornman, K. Chen, B. Douillard, C. P. Lam, D. Anguelov, et al. Multipath++: Efficient information fusion and trajectory aggregation for behavior prediction. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 7814–7821. IEEE, 2022.

[112] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[113] S. Venkatraman. *Latent skill models for offline reinforcement learning*. PhD thesis, Master's thesis, Carnegie Mellon University Pittsburgh, PA, 2023.

[114] L. Wang, J. Liu, H. Shao, W. Wang, R. Chen, Y. Liu, and S. L. Waslander. Efficient reinforcement learning for autonomous driving with parameterized skills and priors. *arXiv preprint arXiv:2305.04412*, 2023.

[115] Z. Wang, A. Novikov, K. Zolna, J. T. Springenberg, S. Reed, B. Shahriari, N. Siegel, J. Merel, C. Gulcehre, N. Heess, and N. de Freitas. Critic regularized regression, 2020.

[116] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez. Learning compositional models of robot skills for task and motion planning. *The International Journal of Robotics Research*, 40(6-7):866–894, 2021.

[117] C. Wen, J. Lin, T. Darrell, D. Jayaraman, and Y. Gao. Fighting copycat agents in behavioral cloning from observation histories. *Advances in Neural Information Processing Systems*, 33:2564–2575, 2020.

[118] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[119] J. Wu, Z. Huang, and C. Lv. Uncertainty-aware model-based reinforcement learning with application to autonomous driving. *arXiv preprint arXiv:2106.12194*, 2021.

[120] Y. Wu, G. Tucker, and O. Nachum. Behavior regularized offline reinforcement learning, 2019.

[121] W. Xi, L. Shi, and G. Cao. An imitation learning method with data augmentation and post processing for planning in autonomous driving. *URL https://opendrivelab.com/e2ead/AD23Challenge/Track_4_pegasus_weitao. pdf*, 2023.

[122] R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, C. Xing, H. Zhang, Y. Lan, L. Wang, and T. Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pages 10524–10533. PMLR, 2020.

[123] D. Xu, Y. Chen, B. Ivanovic, and M. Pavone. Bits: Bi-level imitation for traffic simulation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2929–2936. IEEE, 2023.

[124] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Zou, S. Levine, C. Finn, and T. Ma. Mopo: Model-based offline policy optimization, 2020.

[125] Y. Yuan, X. Weng, Y. Ou, and K. Kitani. Agentformer: Agent-aware transformers for socio-temporal multi-agent forecasting. *arXiv preprint arXiv:2103.14023*, 2021.

[126] W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, and R. Urtasun. End-to-end interpretable neural motion planner. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8660–8669, 2019.

[127] W. Zhan, C. Liu, C.-Y. Chan, and M. Tomizuka. A non-conservatively defensive strategy for urban autonomous driving. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 459–464. IEEE, 2016.

[128] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.