A Prediction and Planning Framework for Scalable Autonomous Driving in Urban Areas

by

Geunseob Oh

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Mechanical Engineering)
in The University of Michigan
2022

Geunseob (GS) Oh

gsoh@umich.edu

ORCID iD: 0000-0002-3575-0376

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Professor Huei Peng, for his continued support and guidance throughout my PhD. I am grateful for his valuable feedback which pushed my thinking and for providing me the freedom to work on a wide range of research directions. Without his support and generosity, I would have not reached the end of this long journey.

I would like to extend my appreciation to my dissertation committee members, Professor Ram Vasudevan, Honglak Lee, and Ilya Kolmanovsky. Beyond their supports for my thesis and constructive feedback, I would like to acknowledge that their works and dedication to the community have always inspired me and motivated me to be a better researcher and communicator.

I would like to thank my collaborators in academia and industry: Jean-Sebastien Valois from Uber ATG, Rahul Goel from Google, Rahul Suresh from Amazon, and David J. Leblanc, Scott Bogard, and Mich Rasulis from UMTRI. I am grateful for these opportunities which provided me first-hand experience of a variety of problems in machine learning and autonomous driving. I would like to give special thanks to Jean-Sebastien for managing my internship at Uber ATG. The experience of working at Uber ATG with you played a pivotal role in my career. I express another special gratitude to Rahul Goel for being a great mentor and supporter at Google. You and your team inspired me to be a stronger developer and researcher.

Many thanks to everyone who supported and encouraged me in and outside of research, including my lab-mates: Nauman, Songan, Xinpeng, Su-Yang, Yuxiao, Boqi, Yuanxin, Minghan, Lu, Shaobing, Pingping, Ding, Ziheng, Juhui, and Steven.

Finally, my greatest appreciation goes to my parents, Donghwan Oh and Eunog Jeon, for all the years of their love and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AAF	Affine Autoregressive Flow
AV	Autonomous Vehicle
BEV	Bird's Eye View
CEM	Cross-Entropy Method
CNN	Convolutional Neural Network
CVAE	Conditional Variational Auto-Encoder
CVAE-H	Conditional Variational Auto-Encoder augmented with Hypernetwork
GAN	Generative Adversarial Network
GMM	Gaussian Mixture Model
IDM	Intelligent Driver Model
HCNAF	Hyper Conditioned Neural Autoregressive Flow
MDN	Mixture Density Network
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSD	Mean Squared Deviation
MPC	Model Predictive Control
NAF	Neural Autoregressive Flow
NLL	Negative Log Likelihood
PDF	Probability Density Function
POM	Probabilistic Occupancy Map
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RS	Random Shooting
SPaT	Signal Phase and Timing
SPMD	Safety Pilot Model Deployment
VAE	Variational Auto-Encoder

ABSTRACT

In the past few years, automotive and technology companies have made major progress towards the real-world deployment of autonomous driving technologies. A few companies have launched fully autonomous driving technologies for taxi rides in small geographic areas. With the initial milestone made, the challenge of developing effective and scalable autonomous driving technologies that can operate in a wide variety of complex urban environments has never been more important.

I propose a prediction and planning framework for self-driving in urban areas to address the challenge. With particular attention to the scalability of the approach, the framework considers unique contexts to each environment and generates effective trajectory plans for different variations of urban driving scenarios in a computationally efficient way. The framework consists of two main tasks: prediction of the environments and planning trajectories of the autonomous vehicle. I mainly leverage learning-based techniques, which have experienced significant progress in recent years, for both prediction and planning tasks.

The prediction task is critical in the framework as accurate predictions of the environment states and their uncertainties are vital to safe and optimal decision-making. To this end, I introduce two powerful conditional generative models, namely HCNAF and CVAE-H, based on normalizing-flow and variational autoencoder, respectively. I show that the two algorithms effectively leverage social and spatial sensor information such as past trajectories of the road-agents and lidar scans of the environment for forecasting the motions of the road agents in a diverse set of environments. I compare my prediction models against state-of-the-art methods using an urban driving dataset and show both methods achieve improved prediction accuracy.

I design the planner to generate near-optimal action sequences autonomously and to consider the uncertainties captured in the prediction outputs. The proposed planner is a model-based random shooting planner with a Gaussian mixture as the backbone distribution. The Gaussian mixture is parameterized using a deep neural network and trained using cross-entropy loss and rewards of the sampled trajectories. Experiments confirm that the proposed planner generates contextual trajectories under various environments in real-time, and the performance compares favorably against several baseline planners, including a dynamic programming planner.

Lastly, I compare the computational efficiency of two different uncertainty representations of the environment. The two representations are (1) trajectory samples of road agents and (2) probabilistic occupancy map, which encodes occupancy probabilities of the road-agents on a continuous 2D heat map. I examine their performances and show that the probabilistic occupancy map representation offers faster and more scalable inference without an excessive sampling of the future states of the road-agents.

CHAPTER I

Introduction

Over the last few years, automotive and technology companies have made major progress towards the real-world deployment of autonomous driving technologies. In late 2020, Waymo, Google's self-driving vehicle company, became the first company that offered commercial autonomous taxi rides to the public in Pheonix, Arizona. More recently in late 2021, Cruise, another American self-driving car company, received a permit from the California Motor Vehicles Department to deploy driverless taxi rides at a max speed of 30 mph between 10 pm and 6 am in the city of San Francisco, California.

With the initial milestone for the deployment made, the challenge of developing effective autonomous driving technologies that can operate in a wide variety of complex urban environments and scale up to different geographic regions has never been more important. As pointed out by a recent publication from Waymo, generalization within and between operating regions is crucial to the overall viability of autonomous driving technologies [1]. Moreover, autonomous vehicles interact with stochastic road users such as human drivers in different driving scenarios and road topologies. The presence of multiple stochastic agents creates uncertainties that grow over time, contributing to the increase in the number of variations of the environments and scenarios that autonomous driving technologies need to handle.

1.1 Urban Driving Environments

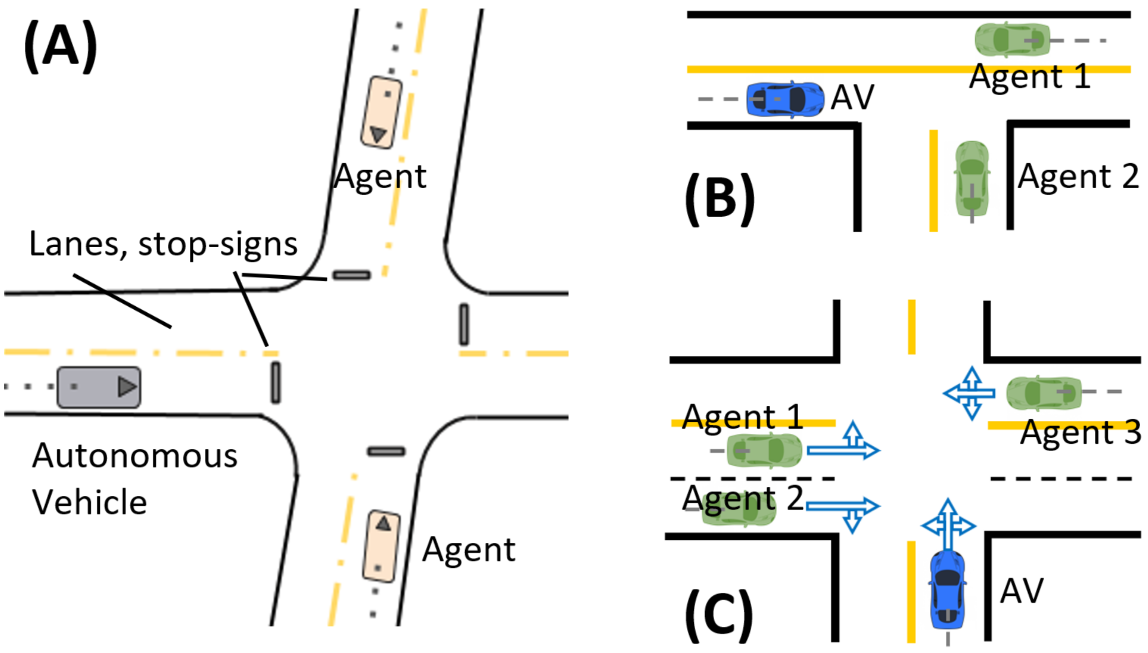


Figure 1.1: (A) Schematics of an urban driving scenario (un-signalized intersection) and (B, C) are two example variations of the scenario. (B) A 3-way intersection where agent 1 can either turn left or go straight, but not turn-right, whereas agent 2 can either go straight or turn left/right. (C) A 4-way scenario with the different number of lanes. The blue arrows indicate available modes unique to each lane. In urban driving environments, AVs need to perform contextual and uncertainty-aware decision-making.

The challenge is especially critical in the decision making in busy urban areas, where an autonomous vehicle (AV) interacts with multiple stochastic road users in diverse environments [2, 3, 4]. For example, at an un-signalized intersection environment as depicted in Figure 1.1, the adjacent human-driven vehicles may slow down, go straight, turn left, or turn right. They may also change their speeds significantly. As the number of road agents increases, the environment gets exponentially complex. Besides, the trajectories may depend on the road topologies (e.g., different shapes, number of lanes, or allowed turns). These variations in the environments require the AV to come up with proper decision-making solutions unique to each environment.

In this regard, a scalable solution means an approach that is capable of considering unique contexts to each environment and generating effective plans adaptively for different scales and variations of the environments without having to significantly increase computational requirements.

1.2 Motivation and Research Objective

Our main interest is to build scalable prediction and planning algorithms for self-driving in urban areas. As the term *scalability* is used throughout the thesis, we first define *scalability* as the ability to autonomously produce *effective* solutions in *computationally efficient* ways for *various* urban driving scenarios. In the following paragraphs, we explain the three keywords of the scalability in detail.

Effectiveness of the solution refers to the quality of predictions and trajectory plans generated by the algorithms. The quality of a prediction model is indicated by the accuracy of the forecasts which can be evaluated using metrics such as mean squared deviation (MSD) of predicted states from the ground-truth states. Another metric is the prediction uncertainty which explains the likelihood of the predicted future states. Access to the prediction uncertainty can be critical for planning in stochastic environments. Other metrics used to measure the quality of predictions are described in Chapter 3 in detail. The effectiveness of a planning model, especially for multi-objective planning problems like ours, is specified by the optimalities of the planned trajectories. The optimality can be quantified using metrics like cumulative rewards of state-action sequences in Reinforcement Learning (RL) or costs associated with the executed trajectories in Optimal Control paradigms.

Computational efficiency represents how much resource an algorithm requires to produce solutions. We quantify the efficiency using both computation time and memory requirement. Computation time is the elapsed time for a model to compute the solutions. Memory requirement means the amount of memory occupied to store the

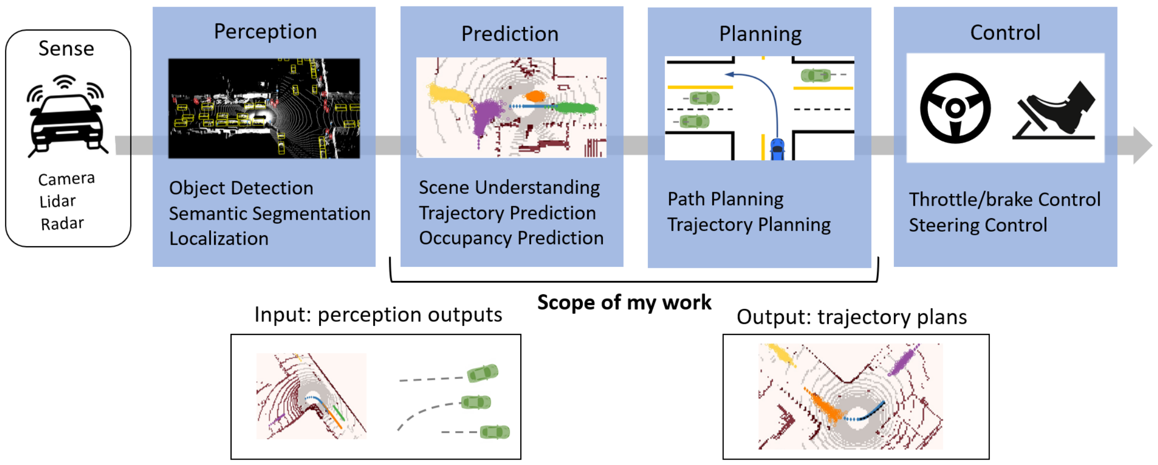


Figure 1.2: A typical autonomy stack of an self-driving car consists of perception, prediction, planning, and control modules. This dissertation focuses on the prediction and planning for autonomous driving. In this regard, the proposed framework takes the perception outputs and produces trajectory plans for downstream modules.

models and solutions. As modern autonomous vehicles continue to employ more sensors and larger artificial intelligence models, managing computation resources is necessary. A computationally in-efficient prediction or planning algorithm could slow down the entire autonomy stack of autonomous driving software, which typically consists of a multitude of modules such as perception, prediction, planning, and control modules.

Generalizability is another key to the scalable self-driving solution. Sometimes referred as robustness in the controls literature, generalizability is the ability to produce effective solutions in *various* environments without needing to manually tune or customize the model. The generalizability is often measured by evaluating the model's performance in unseen environments. As the number of variations of urban driving scenarios is unlimited, the generalizability is essential.

In summary, the main objective of this research is to build a scalable prediction and planning framework for autonomous driving in urban areas in consideration of the three keywords introduced above.

1.3 The Proposed Framework

Figure 1.2 depicts a typical autonomy stack of an AV that consists of perception, prediction, planning, and control modules. The inputs to the AV are sensor information that is obtained using a suite of sensors including camera, lidar, radar, and ultrasonic. The first module of the autonomy stack is the perception module that processes the sensor information to perform tasks such as object detection, semantic segmentation, and localization. The prediction module takes the output of the perception module and predicts a set of future trajectories of road-agents such as human-driven vehicles and pedestrians, or the occupancies of the road-agents. The planning module computes a path and trajectory taking account for the prediction. Lastly, the control module takes the planner output and computes low-level control signals considering various dynamic constraints. This dissertation concerns the prediction and planning modules. In this sense, we assume that the perception outputs are provided and our end product is the trajectory plans to the downstream modules.

Recent data-driven techniques such as modern deep learning models have outperformed various traditional baselines such as heuristics or rule-based methods and adapt to many different environments or variations of the problems. Rule-based approaches typically assume simple interactions with the environment and thus have limited scope and are specific to the intended scenario. Conversely, data-driven approaches can be used to model complex environments with minimal explicit assumptions. The minimal assumptions allow the approach to handle various instances of the problem and thus to scale up better to the diverse urban driving scenarios.

Considering the stochastic and diverse nature of our problem, we mainly use data-driven probabilistic methodologies to design our framework and aim to maximize the performance and generalizability of the solution.

Figure 1.3 portrays the proposed framework which consists of two main problems: prediction of the environments and planning the trajectory of AV. The methodologies

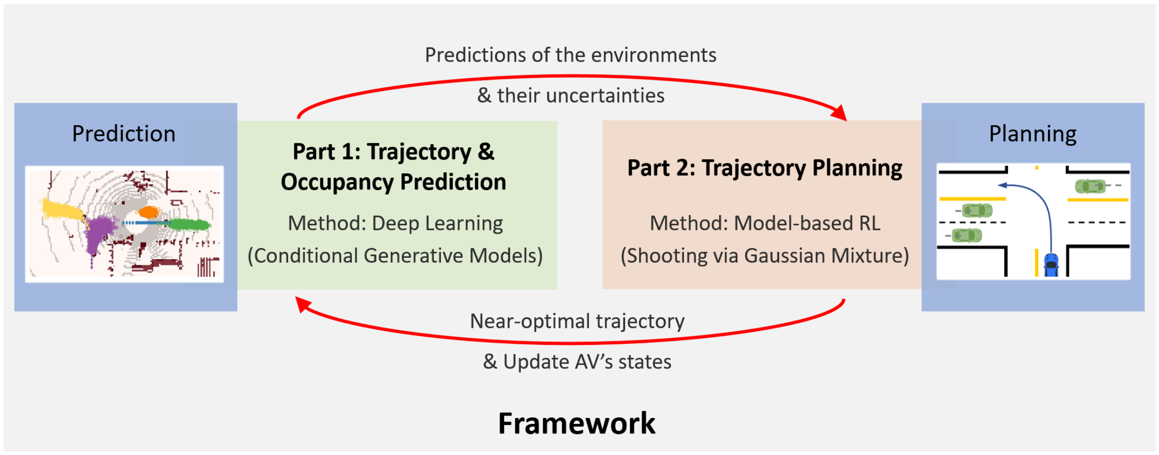

Predictions of the environments
& their uncertainties
Prediction
Part 1: Trajectory & Occupancy Prediction
Method: Deep Learning
(Conditional Generative Models)
Part 2: Trajectory Planning
Method: Model-based RL
(Shooting via Gaussian Mixture)
Planning
Near-optimal trajectory
& Update AV's states
Framework

Figure 1.3: The two parts of the proposed framework. We assume that perception outputs (e.g., labeled past trajectories of the road agents, lidars, and map information) are given to the framework. The framework calls the prediction and planning parts every time new observations become available.

used to tackle each problem mainly comprise of deep learning models, while we also used a fair amount of the optimal control methodologies to design the planner.

We propose a new scalable approach for urban driving scenarios in accordance with the research objective. Our framework aims for effective, generalizable, and computationally efficient predictions and trajectory plannings. To promote the effectiveness and generalizability of the solutions, we paid special attention to the stochasticity of the urban driving environments as we believe that capturing the uncertainties of the environments and using the information for contextual decision-making are the keys.

Our deep-learning forecasting algorithms, HCNAF [5] and CVAE-H [6], effectively model complex probability density functions (pdf) and allow us to predict the future states of the environments and gain access to the uncertainties associated with each prediction. We compare the performance of our prediction models against public benchmark models built on deep generative models such as variational autoencoders, generative adversarial networks, and mixture density networks. Moreover, we verify that the proposed prediction algorithm meets the four desired attributes of the ideal prediction model; probabilistic, multi-modal, context-driven, and general. These attributes are discussed in Chapter 3 in detail.

The subsequent planner takes the prediction outcomes and generates trajectories contextual to the environments for the AV. We model the planner as a mixture of bivariate Gaussian distributions. We optimize the parameters of the distributions (means, co-variances, and mode probabilities) using various deep learning and reinforcement learning techniques to be elaborated in Chapter 4. During the inference, we sample various candidate trajectories from the learned planner and execute the best action sequence for a few steps until new observations arrive. Model predictive control [7] is leveraged to close the loop, that is, each time we have new observations, we repeatedly perform the prediction and planning tasks. We conduct planning experiments to evaluate the proposed planner against a number of baselines including dynamic programming, heuristics, rule-based models, shooting-based methods, and a behavior cloning model. The results support that the proposed planning algorithm meets the three requirements of the ideal planner; near-optimal, general, and computationally efficient.

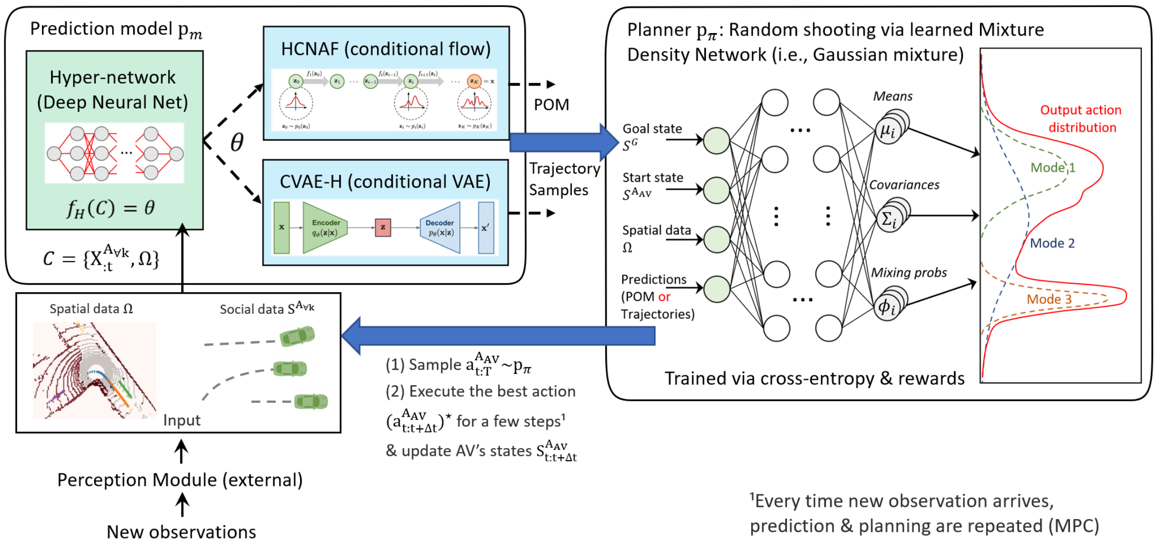

Prediction model p_m
Hyper-network (Deep Neural Net)
$f_H(C) = \theta$
$C = \{X^{A_{\forall k}}_{:t}, \Omega\}$
HCNAF (conditional flow)
CVAE-H (conditional VAE)
θ
POM
Trajectory Samples
Spatial data Ω
Social data $S^{A_{\forall k}}$
Input
Perception Module (external)
New observations
(1) Sample $a^{A_{AV}}_{t:T} \sim p_\pi$
(2) Execute the best action $(a^{A_{AV}}_{t:t+\Delta t})^*$ for a few steps¹
& update AV's states $S^{A_{AV}}_{t:t+\Delta t}$
Planner p_π: Random shooting via learned Mixture Density Network (i.e., Gaussian mixture)
Goal state S^G
Start state $S^{A_{AV}}$
Spatial data Ω
Predictions (POM or Trajectories)
Means μ_i
Covariances Σ_i
Mixing probs ϕ_i
Output action distribution
Mode 1
Mode 2
Mode 3
Trained via cross-entropy & rewards
¹Every time new observation arrives, prediction & planning are repeated (MPC)

Figure 1.4: An overview of the solution network. This figure describes the implementation of the framework presented in Figure 1.3.

Lastly, to reduce inference latency and computational resource requirements, we design the prediction and planning algorithms to be non-autoregressive. Furthermore, our approach employs a unique representation of the uncertainty of the environment called Probabilistic Occupancy Map (POM) which helps reduce the planning time. The details about the non-autoregressive approach and analysis of its theoretical computational complexity are elaborated in Chapter 2.

Figure 1.4 provides an overview of the implementation of the proposed framework. While this section introduced the core ideas of the framework, the theoretical discussion and implementation details are covered in Chapter 3 - *Prediction of the Environment* and 4 - *Planning for Autonomous Driving.*

1.4 Contribution

We propose a novel approach to the prediction and planning for scalable autonomous driving in urban areas. In the following paragraphs, we summarize the main contributions of our work.

The first contribution is the two novel prediction models, HCNAF [5] and CVAE-H [6], which achieve state-of-the-art performance in a public autonomous driving dataset. We confirmed that both HCNAF and CVAE-H outperforms other deep generative model benchmarks in several metrics including minimum of mean squared deviation over k sample predictions (minMSD) and negative log-likelihood (NLL) of the predictions. Furthermore, HCNAF offers access to the exact probability densities and allows trajectory planning with POM.

The second contribution is our planning model, which is capable of generating near-optimal solutions in diverse environments in real-time. The proposed planning model leverages concepts from deep learning to dramatically reduce the computation time and learn to generate near-optimal solutions under various sets of road-agents and geometries.

The last contribution is the proposed framework, which offers a scalable (i.e., effective, generalizable, and computationally efficient) solution for the prediction and planning problems. The scalability of our framework is accredited to the real-time non-autoregressive solution approach and the use of POM as the uncertainty representations, in addition to the highly effective prediction and planning models.

1.5 Organization of the Dissertation

The remainder of the dissertation is organized as follows. We start Chapter 2 by formulating the problem at a high level and discussing two solution approaches that correspond to different uses of the prediction and planning models. Then we analyze their advantages and disadvantages with particular attention to their computational complexities. We explain the reasons behind pursuing the non-autoregressive solution approach. Lastly, we introduce the dataset we used to run and evaluate the proposed framework.

Chapter 3 presents our prediction models. We first elaborate on the four key requirements of effective probabilistic prediction models for autonomous driving in urban areas. We propose two models, namely HCNAF and CVAE-H, that satisfy all of the requirements. A summary of recent prediction models is also provided. Using qualitative examples, we demonstrate that the two proposed algorithms effectively leverage spatio-temporal sensor information and produce accurate predictions of the road agents as well as their uncertainties in diverse environments. We quantitatively compare our prediction models against state-of-the-art learning-based algorithms and confirm that both methods outperform the benchmarks in a public dataset.

Chapter 4 discusses the decision-making task. We begin with detailed descriptions of the planning problem and how the prediction outputs and their uncertainties can be incorporated into the planner. We survey existing planning approaches which utilize optimal control & reinforcement learning and discuss the three key requirements of the scalable planning algorithm. Leveraging insights from recent advances in model-based reinforcement learning algorithms and optimal control, we propose a model-based random shooting planner with a Gaussian mixture as the backbone distribution. The planner is parameterized using a deep neural network and can produce near-optimal trajectory plans in various scenarios in real-time. We conduct experiments to evaluate the proposed planner against a number of baseline planners and show that it outperforms the baselines in terms of solution optimality, generality, and computational efficiency. Furthermore, we present the planning results of the two uncertainty representations of the environment (i.e., trajectory samples and POM).

Chapter 5 is devoted to the summary and discussion of our works. After that, the thesis concludes with an extended discussion about possible future works.

CHAPTER II

Solution Approach

This Chapter extends our discussion on the proposed framework to its mathematical formulation and implementation. Chapter 2 consists of four sections. In Section 2.1, we formulate the problem using Markov decision process. Section 2.2 discusses different implementations of the proposed framework. Specifically, we introduce *Autoregressive* and *Non-autoregressive* implementation and compare their advantages and disadvantages with an emphasis on the computation efficiency. Section 2.3 provides a brief overview of probabilistic occupancy map (POM) and contrasts POM and trajectory samples as the uncertainty representation for the planning of AV with a particular focus on the computation efficiency. Section 2.4 explains the dataset we used to run and evaluate the proposed framework.

2.1 Problem Formulation

To formulate the problem, we use the Markov decision process (MDP), a common framework for decision-making under uncertainties [8, 9, 10, 11, 12, 13]. Specifically, the problem is formulated as an n-th order MDP. In other words, we preserve information about the past states of the environment for n time-steps and leverage them in the prediction. As a result, the state transition is represented as a function of length n tensor with current and previous states and actions. The goal of the autonomous vehicle (AV) is to maximize the expected return under uncertain urban driving environments.

The urban driving environment is assumed to consist of road agents such as human drivers and contextual spatial information such as map and lidar scans. Specifically, we denote the state of a road agent as $S_t^{A_k}$ where the superscript A_k is used to indicate k-th road agent in the scene and the subscript t represents the time-step. $S_t^{A_k} = [X_t^{A_k}, V_t^{A_k}]$ is a collection of positions and speeds of the agent A_k at time t. A superscript and/or subscript can be omitted for the inclusive definition. For example, S_t consists of positions and speeds of all road agents $X_t^{A_{\forall k}}$ and $V_t^{A_{\forall k}}$. Trajectory of a road agent A_k is then defined as $X_{t_1:t_2}^{A_k}$ for $t_1 < t_2$. We assume that all road agents reside in a two dimensional surface (i.e., zero-slope) as it is a common practice that simplifies the problem. Accordingly, $X := [x, y]$ and $V := [\dot{x}, \dot{y}]$ follow the Cartesian coordinate system.

On the other hand, the state of the environment, S_t, includes Ω_t which denotes the spatial features which correspond to contextual static and dynamic scene information extracted from map priors (e.g., lane boundaries and stop-signs) and/or perception modules (e.g., bounding boxes of the road agents overlaid onto a rasterized image of the scene).

$$S_t := [X_t^{A_{\forall k}}, V_t^{A_{\forall k}}, \Omega_t]. \tag{2.1}$$

It is worth noting that our framework assumes that such spatial features are provided by an external perception module or HD map. Ω_t is typically visualized as a Bird's-eye-view (BEV) map.

The action of AV and the k-th road agent at time t are denoted as a_t^{AV} and $a_t^{A_k}$ respectively. The action represents either $a_t := [\ddot{x}_t, \ddot{y}_t]$ or $a_t := [\dot{V}_t, \dot{\theta}_t]$ depending on the dynamic model of the AV. Note that we consider the state transition model for AV f_{AV} to be approximated by a point-mass model with a first-order hold discrete approximation as follows.

$$X_{t+1}^{AV} = f_{AV}(X_t^{AV}, a_t^{AV}) = X_t^{AV} + V_t^{AV}\Delta t + 0.5a_t^{AV}\Delta t^2. \tag{2.2}$$

The proposed prediction model $P_m := P_a(s, s')$ represents the state transition model for the other road agents excluding AV.

$$P_m(X_{t+1}^{A_{\forall k \neq AV}} | X_t^{A_{\forall k}}) := P_a(s, s') = Pr(X_{t+1}^{A_{\forall k \neq AV}} = s' | X_t^{A_{\forall k}} = s, a_t^{AV} = a) \tag{2.3}$$

In summary, the next state of the AV is obtained by a one-step propagation of the *deterministic* system dynamics f_{AV}. In contrast, the next states of other road agents are sampled using the *probabilistic* prediction model P_m.

The expected return (i.e., the cumulative discounted rewards) is denoted as r and obtained by summing over the rewards at each time step r_t. The reward consists of one or more terms related to goal completion, safety, ride comfort, travel time, and in-violations of traffic rules. That is, the planning problem is a multi-objective decision-making problem with the reward function described as in Equation 2.4. The detail of the reward structure is presented in Chapter 4.

$$r := r_{goal} + r_{time} + r_{inviolation} + r_{comfort} + r_{safety} \tag{2.4}$$

Finally, the goal of the planning problem is to obtain the optimal sequence of actions that maximizes the expected return. As described in Equation 2.5, it is a function of states of the environment and actions of the AV from the current time t to the final time T.

$$a_{t:T}^{A_{AV}*} := argmax(E[r_{t:T}(S_{t:T}^{A_{\forall k}}, a_{t:T}^{A_{AV}})]). \tag{2.5}$$

We aim to achieve the goal by building a probabilistic planner P_π with the policy π that generates candidate actions given the state as follows.

$$a_t^{A_{AV}} \sim P_\pi(S_{0:t}, S_{t+1}^{A_{k \neq AV}}). \tag{2.6}$$

2.2 Solution Implementations

This section explains how we tackle the problem defined in the previous section. As our framework consists of two parts: prediction and planning, our solution approaches are also comprised of two main steps; (1) prediction of the future states of the road agents (2) planning optimal action sequences for the AV $a_{t:T}^{A_{AV}}$. Depending on how we structure the two main steps, there could be a multitude of different approaches. Two popular approaches include an end-to-end approach which jointly models and trains the prediction and planning tasks together [14, 15, 16] and a modular approach which separately constructs prediction and planning models as part of the modular pipeline for autonomous driving [17, 18]. A typical modular pipeline consists of localization and mapping, perception, prediction, planning, and vehicle control. In the scope of our problem, the modular approach corresponds to training the prediction model first and using the trained prediction module to generate plausible forecasts of the road-agents in the scene for training and/or conducting inference of the planning model. We pursue the modular approach as the end-to-end approach is less interpretable and more expensive to train.

Two different approaches can be used for the modular approach depending on how the prediction and planning models are employed. In the following paragraphs, we briefly discuss the two approaches, namely *autoregressive* and *non-autoregressive* implementations. An autoregressive implementation indicates that the prediction and planning outputs are produced autoregressively conditional to the previously generated outputs. Conversely, if the prediction and planning outputs of an approach are produced all at once, such an approach is denoted non-autoregressive.

2.2.1 Autoregressive implementation

From t to T, repeatedly perform step 1,2,3:

- Step 1. Prediction of the road agent states : $S^{A_K}_{t+\Delta t} \sim P_m(S^{A_K}_{t+\Delta t} | S_{0:t}),\ \forall k \neq AV$,
- Step 2. Decision-making for AV : $a^{A_{AV}}_t \sim P_\pi(S_{0:t}, S^{A_{\forall k \neq AV}}_{t+\Delta t})$,
- Step 3. State-transition for AV : $S^{A_{AV}}_{t+\Delta t} = f_{AV}(S^{A_{AV}}_t, a^{A_{AV}}_t)$.

The *autoregressive* implementation described above involves repetitions of the prediction (i.e., step 1) and planning steps (i.e., step 2-3) from the current time t to the final time T. Step 1 is denoted as *the prediction task* where the transition function of the environment (i.e., the learned prediction model) is used to predict future states of the road agents. The future states are represented as a conditional distribution of $S^{A_K}_{t+\Delta t}$ given the historical states of the road agents (i.e., $S_{0:t}$). Step 2 describes the decision-making task where we use the learned planner P_π to sample optimal action sequences for the AV, taking into account the historical states of all agents as well as the predicted next states of all road agents from step 1. Step 3 describes a deterministic state-transition for AV which was introduced in Equation 2.2. Once all three steps are performed, and $S^{A_{\forall k}}_{t+\Delta t}$ are obtained, we go back to step 1 and repeat the three steps to obtain $S^{A_{\forall k}}_{t+2\Delta t}$. This iteration is repeated until we obtain $S^{A_{\forall k}}_{t:T}$.

We label this approach as the *autoregressive* approach as the prediction & planning outputs are computed jointly. In other words, step 2 at t requires the state predictions until t and step 1 at $t + \Delta t$ requires the plans until t.

To compute its theoretical time complexity, let us define the following.

- t_{pred}: time to run the prediction model P_m.
- t_{plan}: time to run the planner P_π.
- $t_{transition}$: time to run the AV's state-transition f_{AV}.
- H: planning horizon (= T-t).
- Δt: planning time resolution.
- $T_{autoregressive}$: time to run the autoregressive implementation.
- $T_{non-autoregressive}$: time to run the non-autoregressive implementation.

Since the step 1-3 are repeated H times to obtain the trajectory plans $S_{t:T}^{A_{AV}}$, the computation time complexity for the autoregressive implementation is as follows.

$$T_{autoregressive} = (t_{pred} + t_{plan}) * H/\Delta t, \tag{2.7}$$

Where we omit $t_{transition}$ since the step 3 takes much shorter than step 1 and 2.

Equation 2.7 indicates that the inference time (i.e., time to produce the solution) grows as planning horizon gets longer and time resolution goes finer. This could easily make autonomous driving infeasible for real-world use as autonomous driving in urban area typically requires AV to react quickly to changes in the environments. To take new observations into account, the prediction and planning algorithms are typically employed in the model predictive control (MPC) scheme [7, 13]. This requires $T_{autoregressive}$ to be much shorter than the replanning period t_{replan}.

2.2.2 Non-autoregressive implementation

For a viable solution to autonomous driving in real-time, we propose a *non-autoregressive* implementation. The non-autoregressive implementation is designed to minimize the dependency of the computation time from the planning horizon and time resolution by running the prediction and planning steps only once. The resulting time complexity is then defined as

$$T_{non-autoregressive} = (t_{pred} + t_{plan}). \tag{2.8}$$

The details of the non-autoregressive implementation are as follows.

- Step 1. The prediction : $S^{A_{\forall K \neq AV}}_{t+\Delta t:T} \sim P_m = \prod_{i=t+\Delta t}^{T} P_m(S_i^{A_{\forall k \neq AV}} | S_{0:t})$,
- Step 2. The decision-making : $a^{A_{AV}}_{t:T-\Delta t} \sim P_\pi(S_{0:t}, S^{A_{\forall k \neq AV}}_{t+\Delta t:T})$,
- Step 3. The state-transition : $S^{A_{AV}}_{t+\Delta t:T} = f_{AV}(S_t^{A_{AV}}, a^{A_{AV}}_{t:T-\Delta t})$.

In step 1, the predictions over all time steps are conditionally independent from each other given the observation of the environment $S_{0:t}$. This means that $S_i, t < i <= T$ only depends on $S_{0:t}$ (i.e., state observations up to time t) and does not depend on $S_{t:i}$, hence the name *non-autoregressive*. The road agent state predictions from $t + \Delta t$ to T are generated all at once. In steps 2 and 3, the entire action sequence $a^{A_{AV}}_{t:T-\Delta t}$ is sampled at once and the corresponding trajectory of AV $X^{A_{AV}}_{t+\Delta t:T}$ is computed using $a^{A_{AV}}_{t:T-\Delta t}$.

While the non-autoregressive approach might sacrifice long-term accuracy as it does not consider possible future interactions, the main advantage over the autoregressive approach is computational efficiency. As it only runs the prediction and planning steps exactly once, the inference time is much shorter. As compared in Equation 2.7 and 2.8, the non-autoregressive implementation takes a total of $t_{pred} + t_{plan}$ seconds to run. The autoregressive computation time is $T_{autoregressive} = T_{non-autoregressive} * H/\Delta t$, which means it takes $H/\Delta t$ times longer compared to the non-autoregressive implementation.

As one of our objectives is to construct a scalable algorithm that has high computational efficiency, we implement the non-autoregressive approach.

2.3 Probabilistic Occupancy Map and Time Complexity

The planning time t_{plan} referred in Equation 2.7 and 2.8 can further break down into $t_{plan-generate}$, the time consumed to run the learned planner P_{π} and generate candidate plans, and $t_{plan-select}$, the time to select the most optimal plan among the candidates. While $t_{plan-generate}$ is independent of the predictions $S^{A_{\forall K \neq AV}}$, $t_{plan-select}$ is largely dependent on the predictions. Let us first introduce POM and discuss two different prediction representations before extending the discussion on how each representation impact the computation time.

As briefly mentioned in Section 1.3, POM is one of the representations for the uncertainties in the environment. We define POM as a continuous two-dimensional map where each coordinate $X := (x, y)$ of the map encodes $0 \leq P(X) \leq 1$, the probability density of the coordinate being occupied by road agent(s). Note that $\int_X P(X)dX = 1$ for continuous POMs and $\sum_X P(X) = 1$ for discretized POMs (i.e., occupancy grid map). We visualize POM as heat maps (see the right plot of Figure 2.1) in the vicinity of the AV. The certainty of the prediction is captured as $P(X)$ and multi-modal prediction outcomes are described as the peaks of the probability heat map.

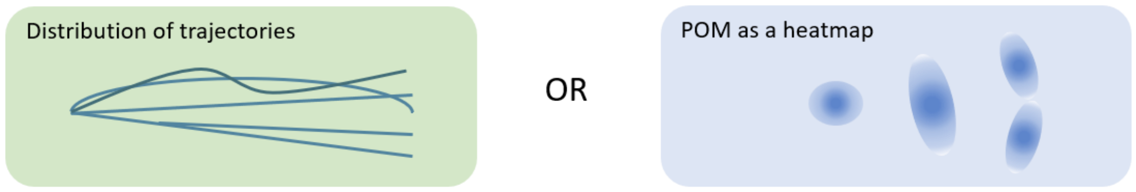


Figure 2.1: Two different uncertainty representations: (left) distribution of trajectories, (right) probabilistic occupancy map.

In the common modular workflow of autonomous driving, planners take the outputs of the prediction modules as inputs to produce trajectory plans. A popular output representation of prediction models is *trajectory*. More precisely, it is either distribution of trajectories or a set of trajectory samples obtained by querying the prediction models. On the other hand, *POM* is a relatively new representation that does not involve the sampling procedure as it records probability into the map. In the following paragraphs, we discuss the advantages and disadvantages of POM compared to the trajectory-based representation.

The main advantage of the trajectory-based representation includes the following. First, it is compatible with various types of planners including optimal controller and RL planners [8, 9, 19, 10, 11]. Second, algorithms that leverage the trajectory representation have achieved great performances for autonomous driving prediction and planning problems. The trajectory representation can also encode the complex and multi-modal nature of road agents in urban driving environments.

However, the major drawback of the trajectory representation is the computational speed. This is elaborated further as follows.

First, a sampled trajectory is just one of all possible future realizations. In most urban driving scenarios, especially at intersections, sampling a few trajectories may not capture all possibilities. Thus, a large number of samples is needed to represent a diverse set of possible futures; however, it is not easy to know how many trajectory samples are enough.

Second, querying the model for a multitude of trajectories may involve heavy computations, which is especially true for large prediction models and models that recursively expand trajectory branches. In practice, a compromise between the prediction accuracy and computation time is made.

Third, it is not guaranteed that the set of samples includes rare events. As the number of trajectory samples grows, the possibility that the set includes rare events increases; however, there is no guarantee. This becomes a severe issue when rare events are safety-critical.

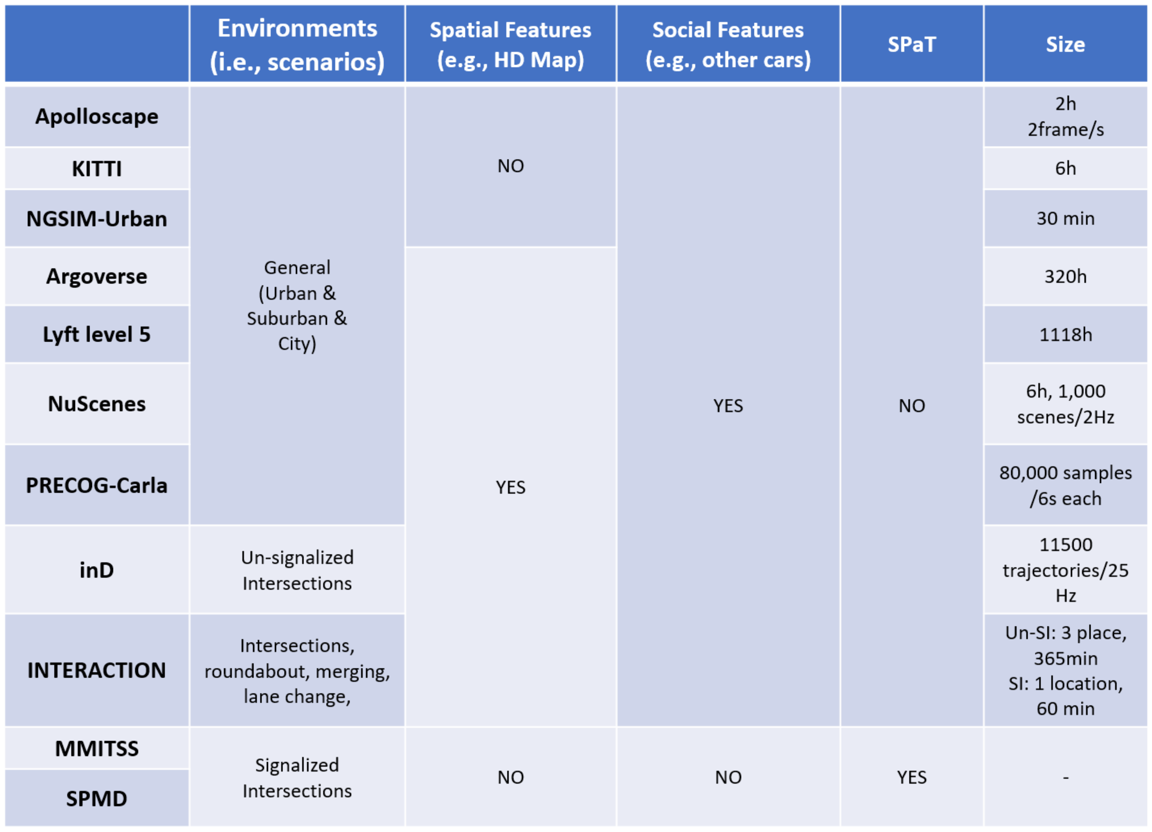
In summary, enough prediction samples should be secured for decision-making to minimize the chance to collide with other road users. Given the number of road agents K in the scene and the number of the prediction samples per road agent $N_{pred}^{A_k}$, the total number of the prediction samples is $\sum_{\forall k} N_{pred}^{A_k}$ which is roughly equal to $K \cdot N_{pred}^{A_k}$. To check the safety of N_{cand} candidate plans against the other road agents, the planner has to run $N_{cand} \cdot K \cdot N_{pred}^{A_k}$ collision checks in total. That is, $t_{plan-select} \sim N_{cand} \cdot K \cdot N_{pred}^{A_k}$ for the trajectory representation.

On the contrary, POM works directly with the uncertainty and depicts a full picture that captures all possible future, including rare events as probabilistic occupancies in the map. Instead of excessive sampling, the planner can simply query the POM, which is the prediction model, for their candidate trajectory coordinates. Therefore, the time complexity for the collision check is N_{cand}, which is a magnitude of $K \cdot N_{pred}^{A_k}$ smaller than the time complexity of the trajectory representation. Since the time complexity of the POM presentation remains constant with respect to the number of sample predictions (i.e., $t_{plan-select} \sim N_{cand}$), it is much more efficient than the trajectory representation. Since POM is relatively new, the challenges are developing an effective prediction algorithm that models POM and designing an effective way of using POM in the planner. These are discussed in Chapter 3 and 4.

2.4 Datasets

Many publicly available large-scale autonomous driving datasets (Argoverse [20], Lyft Level 5 [21], NuScenes [22], Apolloscape [23], KITTI [24], and PRECOG-Carla [25]) include various kinds of urban driving scenarios including lane-keeping driving in straight & curved roads, lane-change, and intersection scenarios. Details of the dataset survey are presented in Figure 2.2.

There also exist scenario-specific datasets that focus on one or two urban driving scenarios such as intersections. Multi-Modal Intelligent Traffic Signal Systems (MMITSS) [26] is a publicly available signalized intersections dataset that includes states of traffic lights (i.e., signal phase and timing, in short, SPaT). There exist a few other private datasets with SPaT, such as Safety pilot model deployment (SPMD) [27]. However, they either are not public or do not contain enough information to estimate the states of the environment. For example, SPMD provides traffic light states but does not provide any information about surrounding vehicles nor geometric data about the environments. INTERACTION [28] and inD [29] are datasets which include un-signalized intersection scenarios, however, they are much smaller compared to the generic datasets.



	Environments (i.e., scenarios)	Spatial Features (e.g., HD Map)	Social Features (e.g., other cars)	SPaT	Size
Apolloscape	General (Urban & Suburban & City)	NO	YES	NO	2h 2frame/s
KITTI	General (Urban & Suburban & City)	NO	YES	NO	6h
NGSIM-Urban	General (Urban & Suburban & City)	NO	YES	NO	30 min
Argoverse	General (Urban & Suburban & City)	YES	YES	NO	320h
Lyft level 5	General (Urban & Suburban & City)	YES	YES	NO	1118h
NuScenes	General (Urban & Suburban & City)	YES	YES	NO	6h, 1,000 scenes/2Hz
PRECOG-Carla	General (Urban & Suburban & City)	YES	YES	NO	80,000 samples /6s each
inD	Un-signalized Intersections	YES	YES	NO	11500 trajectories/25 Hz
INTERACTION	Intersections, roundabout, merging, lane change,	YES	YES	NO	Un-SI: 3 place, 365min SI: 1 location, 60 min
MMITSS	Signalized Intersections	NO	NO	YES	-
SPMD	Signalized Intersections	NO	NO	YES	-

Figure 2.2: Autonomous driving datasets.

As our research objective is a scalable algorithm, we evaluate our approach using various scenarios from the generic datasets rather than the scenario-specific datasets. More specifically, we select PRECOG-Carla, a publicly available dataset created using the open-source Carla simulator for autonomous driving research [25], as our main dataset. The dataset includes up to five road-agents that are human-driven vehicles and processed lidar data from three lidar channels (i.e., two channels above ground and one channel for the ground level inputs). PRECOG-Carla provides several advantages over other generic datasets. First, it provides cleaner processed perception information than other datasets with noisy and imperfect data. This is more suitable to our research as we assume that clean perception information is given to our prediction & planning models. This helps us focus on the modeling and minimize data cleansing which may contribute to performance variances that make it difficult to compare different prediction and planning models. Second, several public benchmarks are available. Recent literature [25, 30, 31, 32, 33, 34, 35, 36] leveraged the PRECOG-Carla dataset to solve the generic autonomous driving prediction problems and provide valuable benchmarks for validating the performance of our models.

CHAPTER III

Prediction of the Environment

Chapter 3 is dedicated to the prediction part of the framework described in the previous chapters. The prediction part spans from taking outputs of perception modules (e.g., labeled trajectories and processed vision data) and predicting future states of other road users to feeding them to the subsequent planner as depicted in Figure 3.1.

In Section 3.4 - 3.8, we present prediction models denoted as p_m in Figure 3.1. In Section 3.9 and 3.10, we report the results of the experiments we conducted to evaluate the performance of the prediction models. Before we elaborate on the proposed prediction models, we first dedicate Section 3.1 - 3.3 to the literature review and explaining the motivation behind the design of our prediction models.

3.1 Introduction

We start the chapter by discussing key characteristics of urban driving environments and the attributes an ideal prediction module should possess.

Autonomous driving in urban areas often involves interactions with stochastic road-agents. When other road-agents exist, AV needs to predict their locations and speeds to avoid collisions. Considering that the road agents behave stochastically, the outputs of the prediction model should not only include bare predictions, but also

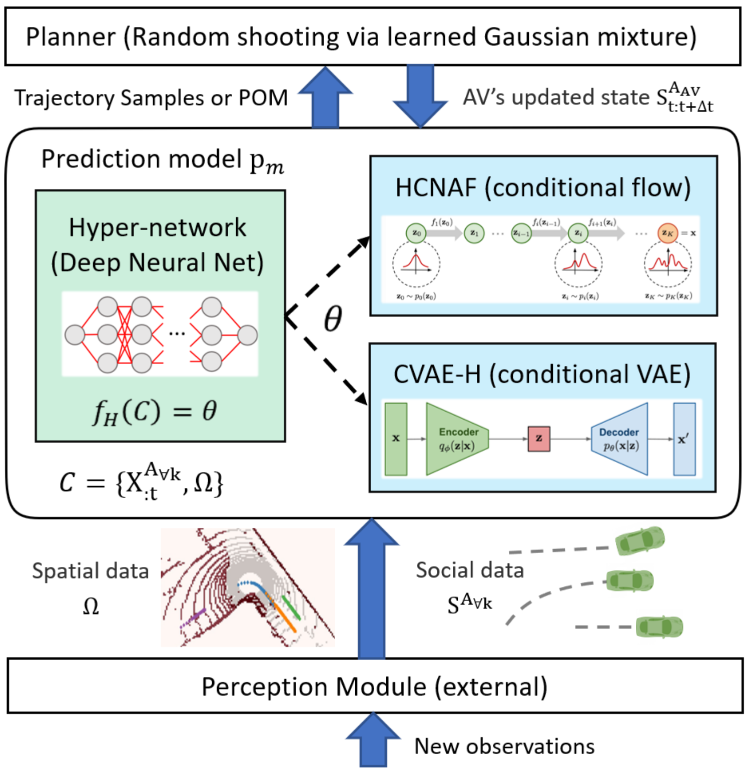

Planner (Random shooting via learned Gaussian mixture)
Trajectory Samples or POM
AV's updated state $S^{A_{AV}}_{t:t+\Delta t}$
Prediction model p_m
Hyper-network (Deep Neural Net)
$f_H(C) = \theta$
$C = \{X^{A_{\forall k}}_{:t}, \Omega\}$
θ
HCNAF (conditional flow)
CVAE-H (conditional VAE)
Encoder $q_\phi(z|x)$
Decoder $p_\theta(x|z)$
Spatial data Ω
Social data $S^{A_{\forall k}}$
Perception Module (external)
New observations

Figure 3.1: An overview of the proposed prediction approach.

their quantified uncertainties.

Another characteristic of urban driving is the diversity of possible trajectories. For example, a vehicle approaching a 4-way intersection has four possible *modes*; go straight, turn left, turn right, or yield if there is a lead vehicle waiting in the queue. Whereas a vehicle approaching a 3-way intersection may not have "go straight" or "turn left" or "turn right" mode available depending on the road geometry. A vehicle approaching a multi-lane intersection may take the inner or outer lane, depending on the destinations and available modes tied to the lanes. A vehicle passing through a crosswalk may slow down or maintain its speed. In other words, there are various factors that contribute to the diversity of the trajectories. Such factors include the relative states of dynamic agents, road geometries, and static obstacles in the scene. To consider these factors, it is important to capture the impact of both social features (e.g., interactions among road-agents) and spatial features (e.g., lanes, stop signs, and crossroads) in the prediction.

Last characteristic of the urban driving environments is that numerous instances of the environments exist. Road configuration alone is diverse; examples include un-signalized intersections that are single-lane, two-lanes, 2-way stops, 4-way stops, u-turns, roundabouts, one-way roads, straight roads with different numbers of lanes, curved roads, and so on. Even environments with similar road configurations sometimes have different allowable motions depending on the number of lanes and angles between adjacent outlets. The presence of static objects and dynamic road-agents diversify the urban environments as well. This means that rule-based or ad-hoc prediction approaches customized to a specific instance of the environment is infeasible because there are just too many variations. A scalable prediction approach should be able to adaptively generate predictions in various instances of the environments.

Taking into account the aforementioned important characteristics, we present four key attributes that an ideal prediction model for autonomous driving in urban environments should exhibit:

1. Probabilistic : can infer uncertainties in the predictions,
2. Multi-modal : can capture diverse and distinct sets of possible futures,
3. Context-driven : can leverage both social and spatial information,
4. General : can produce accurate predictions for new (unseen) inputs.

It is not trivial for a prediction model to satisfy all four requirements. First of all, not all neural-network models and physics-based models are probabilistic unless they build on one of the following approaches. The first group is the approaches that explicitly model prediction uncertainties, such as Mixture Density Network [37], Normalizing-flow models [5, 38, 39, 40], Variational Autoencoder (VAE) [41, 42, 6]). The second group implicitly works with uncertainties (e.g., Generative Adversarial

Networks (GAN) [43]) with aids of probabilistic modules such as Monte-Carlo simulations [44] and kernel density estimation [45]. Note that deterministic models typically make simple assumptions on the dynamics of the environment. The assumptions are either rule-based (e.g., constant acceleration model departing from a stop and constant velocity model in the middle of straight roads) or heuristics (e.g., Intelligent Driver Model (IDM) [46]).

Models that are not multi-modal are deterministic models or uni-modal probabilistic models. Deterministic prediction models usually output *average* behaviors of an agent (i.e., one realization). Although it is possible for them to produce multiple realizations, which may be interpreted as the peaks of multi-modal distributions, the deterministic models are not probabilistic. Uni-modal prediction models may perform well for certain simple scenarios like highway lane-keeping since the vehicles on highways tend to maintain their speeds. However, the uni-modal models are not suitable for urban scenarios like intersections where there exist multiple paths a vehicle can take and the paths are very different from each other (i.e., left-turn, right-turn, drive-straight trajectories).

In order to satisfy the third requirement, the model should be able to reason how the contextual information, such as the social and spatial information impacts the behaviors of road agents. A model that is exclusively based on Convolutional Neural Network (CNN) [47, 48] may fail to capture social features effectively. Similarly, a model that relies solely on social features [49, 50, 32, 51] using Recurrent Neural Network (RNN), attention, or Graph Neural Network (GNN) may not be the best to capture spatial features that are part of HD Maps, images, and/or lidar point clouds. Deterministic models are hardly context-driven as they simplify the predictions by averaging behaviors of road agents and usually do not take into account variations of the social and spatial information as it is infeasible to model all possible realizations of the environment.

Finally, generality, i.e., the ability to work with unseen environments outside the trained environments, is essential to the ideal prediction model. Models that fix the number of output modess such as Gaussian Mixture Model (GMM) [52] is a counter-example. GMM works with the specified number of modes and thus may perform badly when the number of modes changes. Models that leverage scenario-specific heuristics or physical knowledge also typically sacrifice generality over the precision for the targeted scenarios. Relying only on rule-based or heuristic methods requires manual customization efforts towards different scenarios. This is a labor-intensive approach that does not scale well. For this reason, we opt for general prediction models that work and adapt effectively under diverse urban environments.

3.2 Related Works: learning-based prediction models

In this section, we review existing models for the prediction task and categorize them. All models introduced in this section are *generic* prediction models (i.e., the models do not assume a specific environment). It is worth noting that the list is by no means exhaustive, as the field is growing fast.

The first categorization of the prediction approaches is based on input features the prediction model utilizes. We group them into (1) ones that only leverage the social information obtained from trajectories, (2) ones that only utilize the spatial information, which comes from maps and sensors, and (3) ones that use both social and spatial information. As per the third requirement of the ideal prediction model, we look for a model that can leverage both the social and spatial features. In the following paragraphs, we briefly discuss a number of existing models that belong to each category.

Models that focus on the social features include Social-LSTM [49], which uses a pooling layer to capture social interaction of pedestrians, Social-GAN [50], which models multi-modal pedestrian trajectories considering interactions between pedestrians using GAN, and Trafficpredict [32] that uses a graph-based LSTM prediction algorithm to model interactions in heterogeneous traffic scenarios. These models introduce effective ways of modelling interactions between dynamic road agents but do not consider the spatial features.

On the other hand, some models focused on exploiting the spatial features. Examples include IntentNet [47], which introduces a joint detection-prediction model by discretizing the state space and predicting one of eight driving maneuvers. Fast and Furious [48] proposes a deep neural network that jointly solves 3D detection, tracking, and motion forecasting given data captured by a 3D sensor. These models effectively capture underlying traffic rules from the road geometry; however, interactions among road agents are not explicitly modeled.

There are also models that utilize both social and spatial features. DESIRE [30] is a VAE-based multi-modal model that leverages both social interaction and spatial context. SoPhie [53] extends social-GAN approach to take contextual scene information as well as social features. R2P2 [31] takes both social and contextual information and addresses the diversity-precision trade-off of generative forecasting models and formulate a symmetric cross-entropy training objective to address it. PRECOG [25] extends R2P2 and includes both RNNs and CNNs to capture intentions of agents and spatial contexts. MTP [54] uses social and spatial features to model multi-agent interactions and their latent goals into the prediction.

The second categorization is based on the core mechanics of the prediction models as depicted in Figure 3.2, depending on their outputs types, either deterministic or probabilistic. As briefly explained in Section 3.1, probabilistic models provide means to interpret the probability of the prediction, either explicitly or implicitly, whereas deterministic models do not.

Many earlier deterministic models leveraged either physics-based modeling techniques [46] (e.g., constant velocity model, constant acceleration model, car-following

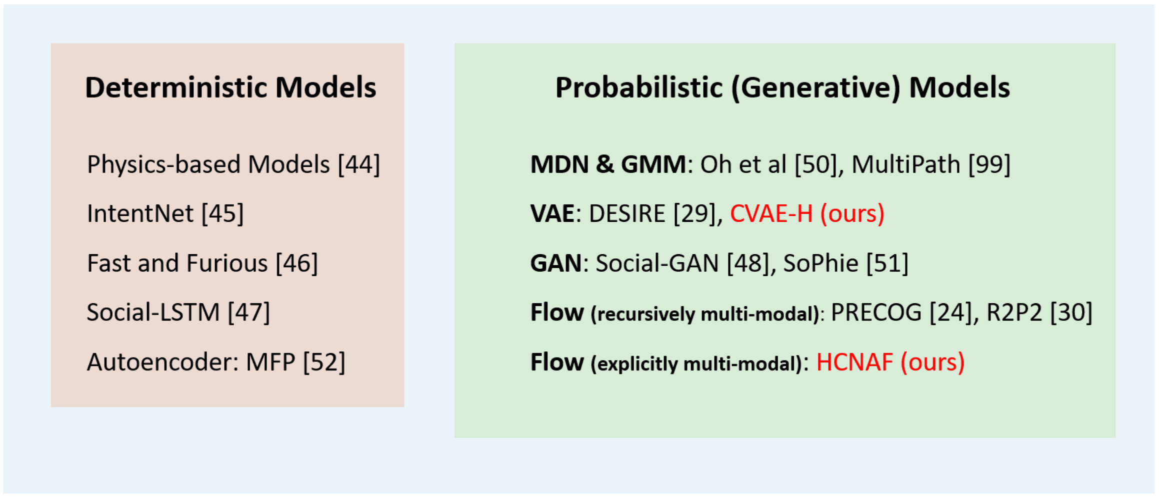

Deterministic Models
Physics-based Models [44]
IntentNet [45]
Fast and Furious [46]
Social-LSTM [47]
Autoencoder: MFP [52]
Probabilistic (Generative) Models
MDN & GMM: Oh et al [50], MultiPath [99]
VAE: DESIRE [29], CVAE-H (ours)
GAN: Social-GAN [48], SoPhie [51]
Flow (recursively multi-modal): PRECOG [24], R2P2 [30]
Flow (explicitly multi-modal): HCNAF (ours)

Figure 3.2: The second categorization is based on whether an algorithm is deterministic or probabilistic (generative).

model) or relatively light-weight neural networks such as multi-layer perceptron (MLP), RNN, and/or CNN [49], while a few recent deterministic models work with larger deep neural networks [47, 48]. As discussed in the previous section, we look for a probabilistic approach that can estimate the uncertainties of its predictions.

Probabilistic models are commonly categorized into discriminative models and generative models [55]. Discriminative models directly approximate $p(x_o|x_i)$ where x_i is the input (i.e., past states of the environment) and x_o is the output prediction. They model $p(x_o|x_i)$ by assuming functional forms of $p(x_o|x_i)$ and learning its parameters. Examples include regression models, support vector machine (SVM), and neural networks that model the conditional distribution $p(x_o|x_i)$ directly. On the other hand, generative models approximate a joint distribution $p(x_o, x_i)$ by assuming functional forms of $p(x_o)$ and $p(x_i|x_o)$ and learning their parameters.

Classical generative models include Bayesian networks and Latent Dirichlet allocation; however, the term *generative models* has been used more broadly after VAE and GAN were introduced. Trained with the principle of maximum likelihood estimation (MLE), VAE and GAN are popular for *generating* the data. Unlike classical generative models, VAE and GAN model a joint distribution $p(x_i)$ using latent variables z where z is not the output of the model, but rather a variable that encodes the underlying dynamics of the data distribution. Furthermore, a group of generative models that are used to model a conditional distribution $p(x_o|x_i)$ via a latent variable z are called *conditional generative models*. Examples include conditional VAE (CVAE) [56], CGAN [57], and both of our prediction models; HCNAF [5] and CVAE-H [6].

Modern generative models, including GAN, VAE, Flow, and Mixture Density Networks (MDN) [37] are popular across several domains of artificial intelligence and machine learning due to their capacities to approximate complex probability distributions. Existing prediction approaches built on these generative models are discussed in the following.

GAN is well-known for producing samples of excellent qualities in computer vision tasks. GAN has also been used in the prediction task for autonomous driving [50, 53, 58]. Although GAN is a probabilistic model, it does not explicitly model the probability [55]. This means that GAN has to rely on expensive Monte-Carlo sampling to approximate the probability distribution of the generated samples to reason the uncertainties of the predictions. As our probabilistic planning approach requires assessment of the uncertainties, GAN hardly qualifies as the prediction model for our problem.

Unlike GAN, VAE is a type of *explicit* generative model that models probability density $p(x)$ explicitly. This property allows us to obtain the uncertainties of the sample predictions directly. Besides, several recent VAE-based models have shown promising results at predicting trajectories of human-driven vehicles in urban environments [30, 54, 42, 6]. Although VAE models data distribution $p(x)$ (i.e., the uncertainty of the prediction sample x) approximately via variational inference [59], the direct and explicit estimation of the uncertainty is a compelling advantage.

Normalizing flow, or Flow, is also a type of *explicit* generative model [55]. Different from VAE, Flow models probability density $p(x)$ explicitly and *exactly.* Normalizing flow began to gain attention in the field of machine learning, especially for density estimation tasks and applications that require predicting uncertainties. It is a new type of explicit generative model that requires meticulous designs to obtain access to the *exact* probability density [31, 25, 5]. Normalizing flow is one of the main components of our prediction models.

MDN is another generative model that obtains *explicit* probability densities. MDN combines conventional neural networks and a mixture of density models. A common choice for the density model is a mixture of Gaussians where the neural network estimates the parameters of the Gaussian mixture, including means, covariances, and mixture ratios. MDN with Gaussian mixtures have been used for prediction tasks [52, 32, 60]. Since MDN works with a pre-determined number of modes, it does not scale well to other environments. For example, an MDN with a mixture of three uni-modal Gaussians trained to predict a vehicle at an intersection scenario where each Gaussian is responsible of 'left-turn', 'right-turn', and 'driving-straight' would not adapt well for a scenario where the vehicle has no forking options.

Figure 3.3 summarizes our discussion in this section by classifying the approaches using the four requirements of the ideal prediction model: probabilistic, context-driven, multi-modal, and general. In short, HCNAF and CVAE-H, the two deep generative models we propose, qualify as the ideal prediction models.

3.3 Related Works: POM

As we utilize the POM representation as an output of the prediction model, relevant literature is reviewed. In the literature, the POM is usually employed in discrete settings as occupancy grid map (OGM). OGM is the *discrete* version of the POM and it has been used in the field of robotics [61, 62] and autonomous driving [63, 64, 65, 66].

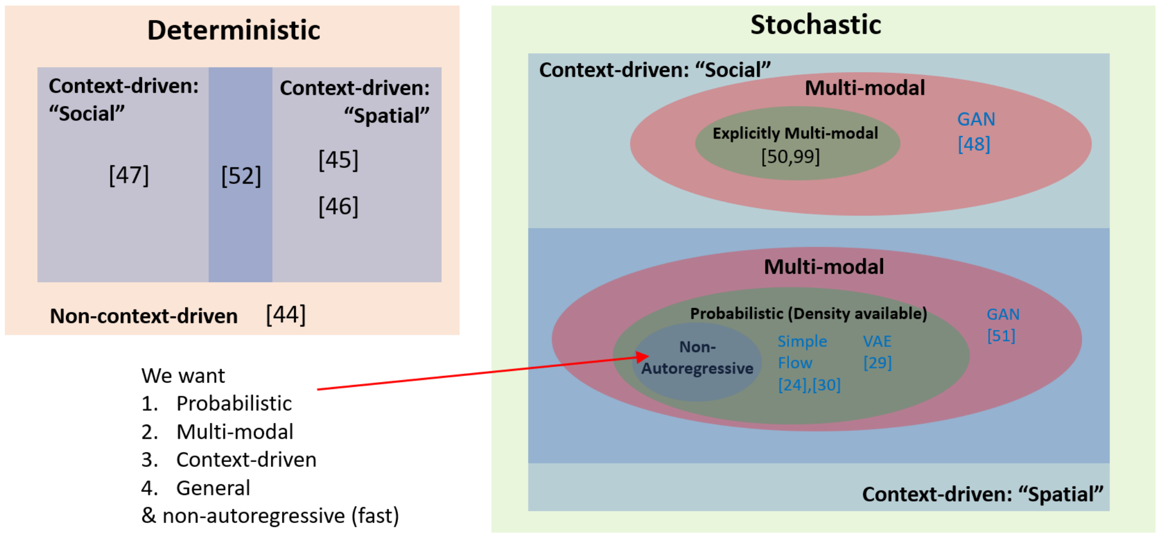


Figure 3.3: The literature reviewed in this section classified based on whether an approach satisfies our requirements (see Section 3.1).

In robotics, the occupancy grid is used to represent the space occupied by objects for planning methods such as Probabilistic Road-Maps (PRM) and Rapidly exploring Random Trees (RRT) to generate collision-free paths.

OGM works under discrete maps as opposed to continuous maps like POM does. In autonomous driving literature with OGM, the prediction problem is usually formulated as a multi-class classification problem with K different grid cells being the classes and the prediction model is trained using cross-entropy over the classes. In [63, 64], an LSTM network was used to produce probabilistic occupancy for discrete grid cells. A softmax layer was used at the end of the network to produce a probability distribution over the finite number of candidate cells. Unlike the two aforementioned studies, [65, 66] worked with multi-dimensional vision data (e.g., map and lidar data). They leveraged encoder-decoder structure combined with RNN as well as convolutional layers to better use the spatial information for the prediction of OGMs.

While the discrete representation of OGM reduces the number of candidate prediction outputs and can decrease the computation cost, we opt for the continuous map representation, POM, for the better prediction accuracy, generalizability of continuous prediction models, and higher flexibility to work with continuous planners.

3.4 Prediction Algorithm Overview

In Section 3.1, we identified the four key attributes of the ideal prediction model for autonomous driving in urban environments. In Section 3.2 and 3.3, we reviewed existing methodologies that have been utilized to solve the prediction task and investigated their capacities. In this section, we provide a high-level description of our prediction algorithms as well as their attributes and explain how they meet all four requirements; probabilistic, multi-modal, context-driven, and general.

Both of our prediction models, HCNAF [5] and CVAE-H [6], consist of two parts, *Explicit density models* and *Hypernetworks* [67]. As introduced in the previous section, explicit density models, more precisely the generative models that explicitly model probability density of the data, satisfy the first attribute of the ideal prediction model by definition (i.e., explicit density models are *probabilistic*). Secondly, our prediction models build on VAE (CVAE-H) and Normalizing Flow (HCNAF) and thus they are capable of approximating arbitrarily complex probability distributions (i.e., our prediction models are *multi-modal*).

While VAE and Flow by definition meet the first and second requirements, vanilla VAE and Flow do not automatically satisfy the third and fourth requirements. The vanilla version of VAE and Flow approximate the data distribution $p(x_i)$ whereas the goal of the prediction task is to obtain conditional probabilities $p(x_o|x_i)$, where x_i and x_o are inputs and outputs of the prediction model. Therefore, a non-trivial modification is required. For VAE, a straight-forward solution is to integrate RNN into the decoder of VAE [68, 69]. However, the same solution does not apply to Flow as RNN violates the Flow's network constraints required to maintain the properties of Normalizing Flow. This is where the second part of our prediction models comes into play.

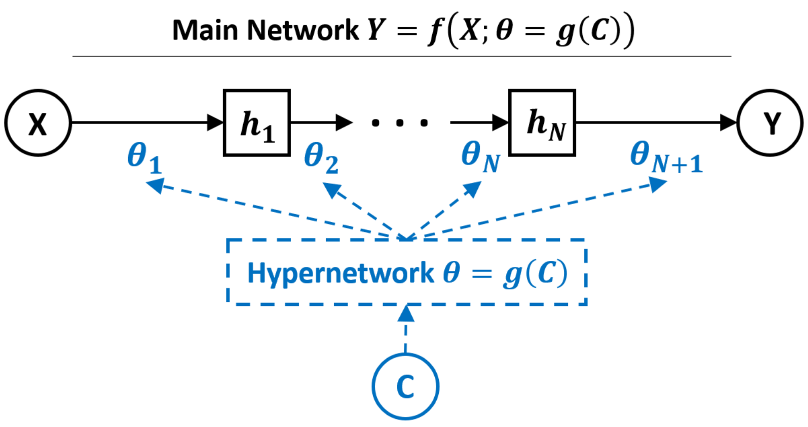

Main Network $Y = f(X; \theta = g(C))$
X
h_1
h_N
Y
θ_1
θ_2
θ_N
θ_{N+1}
Hypernetwork $\theta = g(C)$
C

Figure 3.4: Hyper-network is a neural-network g used to conditionalize another neural-network f (e.g., vanilla VAE and Flow) by providing network parameters of the main neural-network θ.

The second part of our prediction models is Hypernetwork [67] described in Figure 3.4. Hypernetwork provides a way to make the unconditional generative models (e.g., vanilla VAE and Flow) conditional without the use of RNN. Hypernetwork is another neural-network that can consist of various types of neural networks modules such as CNN, RNN, Residual blocks [70] and Attention [71]. One can also use a well-recognized neural network models such as Inception [72], ResNet [73], or BERT [74] as the hypernetwork.

In this sense, hypernetworks take diverse types of data including social and spatial data. The effectiveness of the hypernetworks grows with respect to the capacity of its backbone neural network. Upon use of CNN blocks (spatial features) and RNN/Attention modules (social features), the main neural network becomes *context-driven*.

Furthermore, hypernetworks offer excellent *generalization* capability when it is combined with deep neural networks. This has been demonstrated across diverse tasks in machine learning [67, 75, 76, 77, 78, 79, 80, 81].

We present two prediction models *HCNAF* and *CVAE-H* that are explicit density models combined with hypernetworks. So far, we have elaborated on how both models theoretically satisfy the requirements of the ideal prediction model. Later in this chapter, we provide empirical evidence that our models qualify as the ideal prediction model.

Section 3.6-3.8 are dedicated to the architectural details of our prediction models. But first, we provide theoretical overviews of the deep learning concepts that fabricate our prediction models.

3.5 Preliminaries

In this section, we review the three deep learning concepts that are keys for our prediction models; Variational autoencoder, Normalizing flow, and Hypernetwork.

3.5.1 Variational Autoencoder

Variational Autoencoder [41] is a generative model that learns a data distribution $p(X)$ in an unsupervised way by using an abstract (i.e., latent) variable Z that captures the underlying dynamics of the data distribution X.

As the name suggest, VAE is an *autoencoder* model [82] trained using variational inference [59]. Autoencoder is an unsupervised machine learning model that learns an identity function $f_I = f_{dec}(f_{enc})$ using an encoder f_{enc} and decoder f_{dec} where $f_I(X) = X$. The encoder $f_{enc}(X) = Z$ takes the input data X that are high dimensional and compresses it into a lower dimensional data Z. The decoder $f_{dec}(Z) = X$ then takes Z and recovers the higher dimensional input X. During the reconstruction of the original inputs, the autoencoder aims to discover a more efficient lower dimensional representation of the inputs.

The encoder and decoder of an autoencoder are typically parameterized using neural networks and learned together to minimize the difference between the original and reconstructed inputs. Popular loss functions of *autoencoders* include mean squared deviation (MSD). Note, the loss function for Variational autoencoder is different from that of autoencoders and it is detailed in the following paragraphs.

VAE combines the autoencoder architecture with variational inference and the Bayesian learning techniques. Instead of mapping the input into a fixed latent vector, like the autoencoder, VAE maps the input into a probability distribution. Accordingly, the encoder and decoder are no longer deterministic. Instead, the encoder and decoder each models the probability distributions $p(Z|X)$ and $p(X|Z)$. As the compression $X \rightarrow Z$ and generation $Z \rightarrow X$ steps become probabilistic, VAE can be used for generative tasks by sampling Z and passing it through the decoder to reconstruct (or predict) X.

Let us denote $p(Z|X)$ and $p_m(Z|X)$ as the true posterior and model posterior distributions. Likewise, $p(X|Z)$ and $p_m(X|Z)$ are denoted as the true likelihood and model likelihood. Finally, we denote $p(Z)$ as the prior distribution. With the introduction of probability distributions, the learning objective becomes maximizing the likelihood of the training data:

$$\theta^* = argmax(\prod_{k=1}^{N} p_m(X_k)), \tag{3.1}$$

where θ^* represents the optimal parameters for the encoder and decoder. Using Bayes rule and the law of total probability, $p(X_k)$ can be expressed follows:

$$p(X_k) = \int_Z p(X_k, Z)dZ = \int_Z p(X_k|Z)p(Z)dZ. \tag{3.2}$$

Equation 3.2 is intractable as the integration over the entire domain of the continuous random variable Z is very expensive. This is where the variational inference [59] comes into play. Instead of directly maximizing the likelihood, we could maximize a computable term called *Evidence Lower BOund (ELBO)* of the likelihood. As the name indicates, ELBO is a lower bound of the original likelihood, i.e., $p(X) \geq ELBO$. By using Bayes rule and probability lemmas, we can express $p(X)$ using ELBO and

KL divergence as follows.

$$\begin{aligned}
log(p(X)) &= \int_Z p_m(Z|X_k) log(p(X_k))dZ, \\
&= \mathbb{E}_{Z\sim p_m(Z|X_k)}\big[log(p(X_k))\big], \\
&= \mathbb{E}_{Z\sim p_m(Z|X_k)}\Bigg[log(\frac{p(X_k|Z)p(Z)}{p(Z|X_k)})\Bigg], \\
&= \mathbb{E}_{Z\sim p_m(Z|X_k)}\Bigg[log(\frac{p(X_k|Z)p(Z)}{p(Z|X_k)}\frac{p_m(Z|X_k)}{p_m(Z|X_k)})\Bigg], \\
&= \mathbb{E}_Z\Bigg[log(p(X_k|Z))\Bigg] - \mathbb{E}_Z\Bigg[log(\frac{p_m(Z|X_k)}{p(Z)})\Bigg] + \mathbb{E}_Z\Bigg[log(\frac{p_m(Z|X_k)}{p(Z|X_k)})\Bigg], \\
&= \mathbb{E}_Z\Bigg[log(p(X_k|Z))\Bigg] - KL(p_m(Z|X_k)||p(Z)) + KL(p_m(Z|X_k)||p(Z|X_k)), \\
&= ELBO + KL(p_m(Z|X_k)||p(Z|X_k)),
\end{aligned} \tag{3.3}$$

where ELBO is defined as $\mathbb{E}_Z\Big[log(p(X_k|Z))\Big] - KL(p_m(Z|X_k)||p(Z))$. This corresponds to a summation of reconstruction error and negative KL divergence between the posterior and prior distributions.

Since KL divergence is always non-negative (i.e., $KL(\cdot) \geq 0$), Equation 3.3 is equivalent to $log(p(x)) \geq$ ELBO. The ELBO is easily computable as long as the posterior, prior, and generative distributions are modeled using explicit density models. It is a tractable approximation of the original likelihood. Then, the learning objective of VAE and optimal parameters of the network are expressed as follows.

$$\begin{aligned}
L_{VAE}(\theta) &= max(ELBO(\theta)). \\
\theta^* = argmin_\theta\Big[&- \mathbb{E}_Z\big[log(p_m(X_k|Z;\theta))\big] + KL(p_m(Z|X_k;\theta)||p(Z))\Big].
\end{aligned} \tag{3.4}$$

Note that the expectation term in the loss function assumes that Z is sampled using $p_m(Z|X_k;\theta)$. However, the sampling process does not allow a back-propagation of the gradient as it is a stochastic process. In order to merge the sampling process into a gradient-based learning scheme (i.e. optimizing parameters of neural networks by descending the gradient of the learning objective), the *reparameterization* trick [41] is used, which introduces another random variable (i.e., noise) $\epsilon \sim N(0, \mathrm{I})$. This trick states that Z follows a normal distribution whose mean and standard deviation are the outputs of the encoder; $Z = \mu(X) + \sigma(X) \odot \epsilon$, where $\odot$ denotes element-wise multiplication. This formulation enables the back-propagation of the gradient as μ and σ are learnable terms that bridge the gradient flow from the encoder to the decoder.

3.5.2 Normalizing Flow

Flow, or Normalizing flow, is a type of deep generative models and explicit density models. Flow is also known as invertible (bijective) neural networks. Flow-based models learn data distribution via the principle of maximum likelihood [55] so as to generate new data and/or estimate the likelihood of a target distribution.

Normalizing flow transforms a probability density function into another probability density function (pdf) via the change of variable theorem. Assume that we would like to estimate an unknown target distribution $p(x)$. Normalizing flow obtains $p(X)$ using another distribution $p(z)$ whose pdf is known (e.g., Gaussian distribution) by constructing an invertible function $f(z) = x$ between the latent variable z and target variable x.

Once such function f is constructed, then the relationship denoted as *change of variable theorem* holds between the two probability distributions $p(x)$ and $p(z)$.

For univariate variables $x \in R^1$ and $z \in R^1$, the change of variable formula is as follows.

$$p(x) = p(z)\left|\frac{dz}{dx}\right|, \tag{3.5}$$

which is drawn by differentiating the following probability density definition with respect to x:

$$\int p(x)dx = \int p(z)dz = 1. \tag{3.6}$$

The equation 3.5 states that one can infer $p(x)$ using a known probability density function (pdf) $\pi(z)$ and the derivative of the invertible function $z = f^{-1}(x)$.

The multivariate extension ($x \in R^D$ and $z \in R^D$) is similar, except $\frac{dz}{dx}$ is replaced by the determinant of the Jacobian of the function f as follows:

$$p(x) = p(z)\left|det\frac{dz}{dx}\right| = \pi(f^{-1}(x))\left|det\frac{df^{-1}(x)}{dx}\right|. \tag{3.7}$$

The invertible function $f(z) = x$ may consist of a sequence of invertible transformations $f = f_1 \circ f_2 \circ ... \circ f_N$ where each invertible transformation f_k typically is modelled using affine coupling layers [83, 38, 84, 85] or linear layers (i.e., MLP) [39, 40, 86, 5] followed by an invertible activation function (e.g., exponential function, ReLU, tanh).

In addition to access to the exact probability density, Flow offers data generation capability by sampling latent variables $z \sim \pi()$ and passing it through f. As the accuracy of the approximation $f(z) = x$ increases, the modeled pdf $p_{model}(x)$ converges to the true $p(x)$ and the quality of the generated samples improves.

In contrast to other classes of deep generative models such as VAE [41] and GAN [43], Flow has the following unique properties:

1. Computation of an **exact** probability is essential in the uncertainty-aware decision making for autonomous driving, especially for POM-based approaches. In contrast, VAE infers $p(x)$ using ELBO, which is an estimate for the lower

bound of $p(x)$. It is unclear how well ELBO actually approximates $p(x)$ and how ELBO can be utilized for tasks that require exact inference. While GAN was found to synthesize high-quality samples across different machine learning and computer vision domains, obtaining the density estimation and/or probability computation for the generated samples is non-trivial.

2. The expressivity of Flow-based models allows the models to capture complex data distributions. A recently published AF model called *Neural Autoregressive Flow (NAF)* [40] unified earlier AF models including [87, 39] by generalizing their affine transformations to arbitrarily complex non-linear monotonic transformations. Conversely, the vanilla VAE uses *unimodal* Gaussians for the prior and the posterior distributions. In order to increase the expressivity of VAE, some have introduced more expressive priors [88] and posteriors [89, 87, 90, 91, 92, 86] that leverage Flow.

The class of invertible neural-net based autoregressive Flows, including NAF [40] and $BNAF$ [86], can approximate rich families of continuous pdfs. However, NAF and $BNAF$ do not handle *external* conditions (e.g. classes in the context of GAN vs cGAN [57]). In other words, those models are designed to compute $p(x_t)$ conditioned on previous inputs $x_{1:t-1}$ autoregressively to formulate $p(x_t|x_{1:t-1})$. This formulation is not suitable for taking arbitrary conditions, except the autoregressive ones. This limits the extension of NAF to applications that work with conditional probabilities $p(X|C)$, such as the prediction task for autonomous driving.

$$cMAF : x_d = \mu(x_{1:d-1}, C) + \sigma(x_{1:d-1}, C)z_d, \tag{3.8}$$

MAF and $cMAF$ were proposed in [39] to model affine Flow transformations with and without additional external conditions. As shown in Equation 3.8, the transformation between z_d and x_d is affine and the influence of C over the transformation relies on μ, σ, and stacking multiple Flows. These may limit the contributions of C to the transformation. This explains the need for a conditional autoregressive Flow that does not have such an expressivity bottleneck.

Other flavor of Normalizing flow methods builds upon invertible convolutions such as 1×1 in (Glow) [84] and $d \times d$ in [93]. The work in [85] modified Glow to work with external conditions for structured output learning, yielding a non-autoregressive Normalizing flow model.

We resolve the aforementioned limitation of the autoregressive Flow models (e.g., *NAF* and *BNAF*) by integrating autoregressive Flow with a hypernetwork. Before presenting the details of our solutions, we finish reviewing the hypernetwork in the following subsection.

3.5.3 Hypernetwork

Assume a function that takes two inputs X, C and outputs Y, i.e., $f : (X, C) \rightarrow Y$. Given that f is a probabilistic mapping, $p(Y|X, C)$ represents a probability distribution of the output Y conditioned on the two inputs. Learning such f and the corresponding conditional distribution can be achieved using two different approaches; (1) the embedding and (2) the hypernetwork approach, as depicted in Figure 3.5.

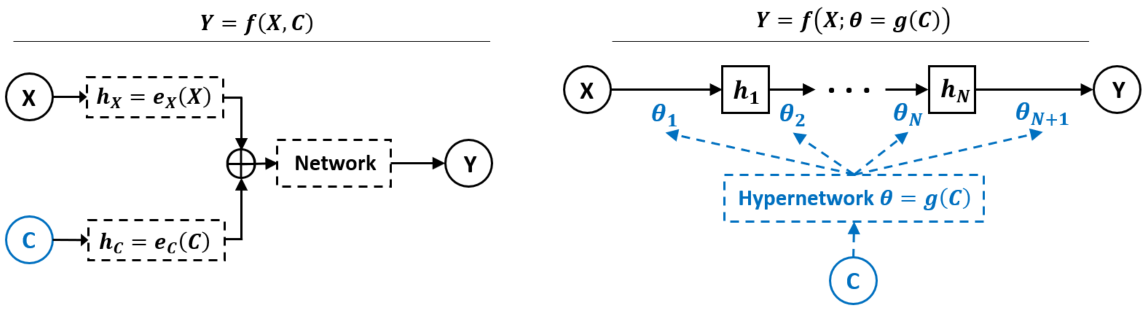


Figure 3.5: Two approaches to model conditional distributions. The embedding approach on the left and hypernetwork approach on the right are depicted.

As commonly used in machine learning, the embedding approach is a straightforward way of learning f. This approach implies a model that consists of the embedding modules $e_1(X) = h(X)$ and $e_2(C) = h(C)$ where the two outputs are concatenated $h := h(X) \oplus h(C)$ and passed to another network $f(h) = Y$.

The hypernetwork approach [67] is a less intuitive solution that relies on a hierarchical structure between two models; one called a primary network g produces weights of the other separate network f. This corresponds to $f(X; \theta) = Y$ where $\theta = g(C)$. Observe that the embedding approach uses an embedding network e and treats C equally to X as part of the input. On the other hand, the hypernetwork approach models the conditional mapping $f_C(X) = Y$, i.e., learning the contributions of X to Y via a parameterized function $f(X; \theta)$ where the parameters themselves are the outputs of another network g, which is independent of f.

The ability to effectively learn conditional functions (i.e., model a function that transforms into different functions depending on the condition C) corresponding to the property of modularity [81] is a major benefit of the hypernetwork approach. A recent study [81] showed that hypernetwork exhibits the property of modularity for a large g and small f. Furthermore, the hypernetwork approach uses much smaller networks compared to the embedding approach to achieve the same performance in modeling conditional distributions.

3.6 HCNAF, A Flow Parametrized Using Hypernetwork

In Section 3.4, a high level description about HCNAF and CVAE-H was provided. Section 3.6 and 3.7 extend Section 3.4 by presenting the mathematical and architectural details of HCNAF and CVAE-H. In Section 3.8, we elaborate on how we leverage and customize the two models for solving the prediction problem.

HCNAF (Hyper Conditioned Neural Autoregressive Flow) [5] is a flow-based conditional generative model. Like other Normalizing flow models, HCNAF leverage the change of variable theorem (see Equation 3.5 and 3.7) to construct a transformation between two random variables $X = [x_1, x_2, ..., x_D] \in \mathbb{R}^D$ and $Z = [z_1, z_2, ..., z_D] \in \mathbb{R}^D$. The transformation is modeled using an invertible neural network $f(X;\theta) = Z$ parameterized with θ. The following relationship is then obtained between the probability densities of the two random variables:

$$p(X|C) = p(Z|C) \left| \det \frac{dZ}{dX} \right| = N_{pdf}(f(X;\theta); 0, I) \left| \det \frac{df(X;\theta)}{dX} \right|, \tag{3.9}$$

where Z is a random variable drawn from a standard multivariate normal distribution and N_{pdf} denotes the pdf of Z.

The main difference from other flow-based models is the way HCNAF is conditionalized. Specifically, the parameters θ are determined by arbitrarily complex conditions $C \in \mathbb{R}^{D_c}$ via a separate neural network $f_H(C) = \theta$ previously introduced as *Hyper-network*. HCNAF models a conditional joint distribution $p(x_1, x_2, ..., x_D|C)$ autoregressively on $x_{1:D}$, by factorizing it over D conditional distributions $\prod_{d=1}^{D} p(x_d|x_{1:d-1}, C)$.

Among various normalizing flow models, *NAF* [40] model is leveraged in HCNAF. NAF achieved state-of-the-art performances in various density estimation tasks; however, the vanilla NAF is un-conditional and unable to work with the complex conditions C. HCNAF combines Hyper-network with NAF to make the resulting flow conditional, hence the name HCNAF (Hyper Conditioned NAF).

NAF [40] and HCNAF both use neural networks but are different in their probability modeling, conditioner network structure, and Flow transformation as specified below:

$$\left.\begin{aligned} p(x_1, x_2, ..., x_D) = \prod_{d=1}^{D} p(x_d|x_{1:d-1}), \\ f_c(x_{1:d-1}) = \theta_d, \\ f(x_d;\theta_d) = z_d, \end{aligned}\right\} \quad \text{NAF} \tag{3.10}$$

$$\left.\begin{aligned} p(x_1, x_2, ..., x_D|C) = \prod_{d=1}^{D} p(x_d|x_{1:d-1}, C), \\ f_H(C) = \theta, \theta_d \in \theta, \\ f(x_d; x_{1:d-1}, \theta_d) = z_d. \end{aligned}\right\} \quad \text{HCNAF} \tag{3.11}$$

In Equations 3.10, *NAF* uses a conditioner network f_c to obtain the parameters θ_d for the transformation between x_d and z_d, which is parameterized by autoregressive conditions $x_{1:d-1}$. In contrast, in Equations 3.11, HCNAF models the transformation to be parameterized on both $x_{1:d-1}$, and an arbitrarily large external conditions C in non-autoregressive fashion via the hypernetwork f_H. For probability modeling, the difference between the two is analogous to the difference between VAE [41] and conditional VAE [56], and that between GAN [43] and conditional GAN [57].

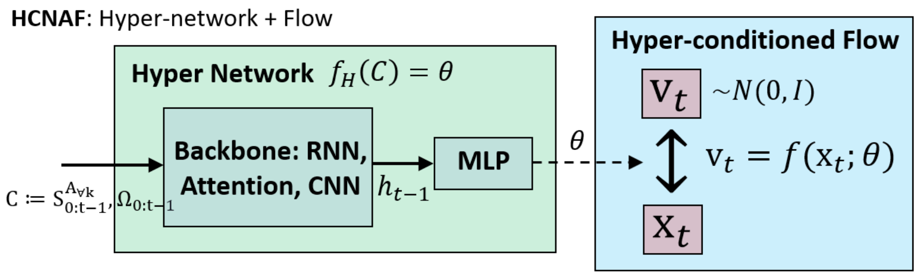

HCNAF: Hyper-network + Flow
Hyper Network $f_H(C) = \theta$
Backbone: RNN, Attention, CNN
MLP
h_{t-1}
$C := S^{A_{\forall k}}_{0:t-1}, \Omega_{0:t-1}$
θ
Hyper-conditioned Flow
V_t $\sim N(0, I)$
$v_t = f(x_t; \theta)$
X_t

Figure 3.6: Schematic of HCNAF

As illustrated in Figure 3.6, HCNAF consists of two modules: 1) a neural-net based conditional autoregressive Flow (i.e., Hyper-conditioned Flow), and 2) a hypernetwork that computes the parameters of 1). The modules are detailed in the following subsections.

3.6.1 Hyper-conditioned Flow

The proposed conditional AF is a bijective neural-network $f(X; \theta) = Z$, which models transformation between random variables X and latent variables Z. The network parameters $\theta := [\mathbf{W}, \mathbf{B}]$ are determined by the hyper-network $f_H(C) = \theta$.

The main difference between regular neural nets and Flow models is the invertibility of $f^{-1}(Z) = X$ as regular networks are not typically invertible.

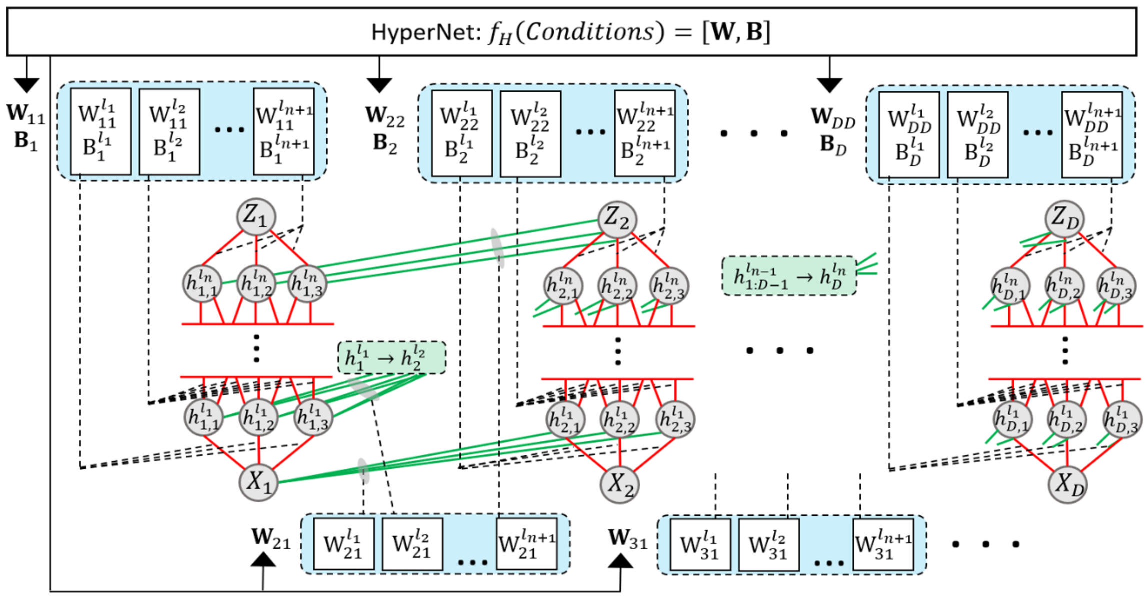

HyperNet: $f_H(Conditions) = [\mathbf{W}, \mathbf{B}]$
$\mathbf{W}_{11}$ $\mathbf{B}_1$
$\mathbf{W}_{22}$ $\mathbf{B}_2$
$\mathbf{W}_{DD}$ $\mathbf{B}_D$
$\mathbf{W}_{21}$
$\mathbf{W}_{31}$
$h_1^{l_1} \rightarrow h_2^{l_2}$
$h_{1:D-1}^{l_{n-1}} \rightarrow h_D^{l_n}$

Figure 3.7: HCNAF's conditional AF model f (i.e., Hyper-conditioned Flow) is a neural-net whose parameters are determined by a hypernetwork f_H.

The detailed architecture of the hyper-conditioned flow is shown in Figure 3.7. The figure describes a D dimensional conditional AF with n hidden layers with 3 nodes. The dash lines refer to connections from f_H to *parameters* of f. In each dimension d of the Flow, the bijective transformation between x_d and z_d are modeled with a MLP with n hidden layers as follows:

$$x_d \leftrightarrow h_d^{l_1} \leftrightarrow h_d^{l_2} \leftrightarrow ... \leftrightarrow h_d^{l_n} \leftrightarrow z_d(= h_d^{l_{n+1}}). \tag{3.12}$$

The connection between two adjacent hidden layers $h_d^{l_k}$ and $h_d^{l_{k-1}}$ is defined as:

$$h_d^{l_k} = \phi(W_{dd}^{l_k} h_d^{l_{k-1}} + \sum_{r=1}^{d-1}(W_{dr}^{l_k} h_r^{l_{k-1}}) + B_d^{l_k}), \tag{3.13}$$

where subscript and superscript each denotes Flow number and layer number. Specifically, $h_d^{l_k}$ is the hidden layer l_k of the d-th Flow. $W_{dr}^{l_k}$ and $B_d^{l_k}$ denote the weight matrix that defines contributions to the hidden layer l_k of the d-th Flow from the hidden layer l_{k-1} of the r-th Flow, and the bias matrix that defines the contributions to the hidden layer l_k of the r-th Flow. Finally, $\phi()$ is an activation function.

In Figure 3.7, red lines between adjacent hidden layers $h_d^{l_{k-1}}$, $h_d^{l_k}$ ($\forall d, 1 \leq k \leq n{+}1$) indicate that $W_{dd}^{l_k}$ is strictly positive. Green lines between layers $h_a^{l_{k-1}}$, $h_b^{l_k}$ in different Flow dimensions ($1 \leq a < b \leq D, 1 \leq k \leq n+1$) have no such constraint (i.e., $W_{ba}^{l_k}$ is unconstrained.)

The connection between x_d and the first hidden layer, and between the last hidden layer and z_d are defined as:

$$\begin{aligned} h_d^{l_1} &= \phi(W_{dd}^{l_1} x_d + \sum_{r=1}^{d-1}(W_{dr}^{l_1} x_r) + B_d^{l_1}), \\ z_d &= W_{dd}^{l_{n+1}} h_d^{l_n} + \sum_{r=1}^{d-1}(W_{dr}^{l_{n+1}} h_r^{l_n}) + B_d^{l_{n+1}}. \end{aligned} \tag{3.14}$$

h^{l_k} are the hidden units at the hidden layer l_k across all Flow dimensions $d = 1 : D$ and are expressed as:

$$h^{l_k} = \phi(W^{l_k} h^{l_{k-1}} + B^{l_k}), \tag{3.15}$$

where W^{l_k} and B^{l_k} are the weights and biases matrices at the hidden layer l_k across all Flow dimensions:

$$W^{l_k} = \begin{bmatrix} W_{11}^{l_k} & \mathbf{0} & \dots & \mathbf{0} \\ W_{21}^{l_k} & W_{22}^{l_k} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ W_{D1}^{l_k} & W_{D2}^{l_k} & \dots & W_{DD}^{l_k} \end{bmatrix}, \quad B^{l_k} = \begin{bmatrix} B_1^{l_k} \\ B_2^{l_k} \\ \vdots \\ B_D^{l_k} \end{bmatrix}. \tag{3.16}$$

Likewise, $\mathbf{W}$ and $\mathbf{B}$ denote the weights and biases matrices for all Flow dimensions across all the layers. Specifically, $\mathbf{W} := \{\forall k : W^{l_k}\}$ and $\mathbf{B} := \{\forall k : B^{l_k}\}$.

Finally, $Z = f(X)$ is obtained by computing the terms from Equation 3.15 for all the network layers, from the first $X = h^{l_0}$ to the last layer, $Z = h^{l_{n+1}}$.

We designed HCNAF so that the hidden layer units $h^{l_k}_{1:D}$ are connected to the hidden units of previous layers $h^{l_{k-1}}_{1:D}$, inspired by *BNAF*, as opposed to taking $h^{l_{0:n+1}}_{d}$ as inputs to a separate hyper-network to produce $h^{l_{0:n+1}}_{d+1}$ over $d = 1 : D$, such as presented in *NAF*. This approach avoids running the hyper-network D times; an expensive operation for large hyper-networks. By designing the hyper-network to output $h^{l_{0:n+1}}_{1:D}$ all at once, we reduce the computation load, while allowing the hidden states across all layers and all dimensions to contribute to the Flow transformation, as x_d is conditioned not only on $x_{1:d-1}$, but also on all the hidden layers $h^{l_{0:n+1}}_{1:d-1}$.

All Flow models must satisfy the following two properties to ensure the invertibility and efficient computations: 1) monotonicity of $f(X) = Z$ for the invertibility, and 2) tractable computation of the jacobian matrix determinant $\left| det \frac{dZ}{dX} \right|$.

3.6.2 Invertibility of the Autoregressive Flow

The monotonicity requirement is equivalent to having $\forall d : \frac{dz_d}{dx_d} > 0$, which is further factorized as:

$$\frac{dz_d}{dx_d} = \frac{dz_d}{dh^{l_n}_d} \prod_{k=1}^{n-1} \frac{dh^{l_{k+1}}_d}{dh^{l_k}_d} \frac{dh^{l_1}_d}{dx_d} = W^{l_{n+1}}_{dd} \prod_{k=0}^{n-1} \frac{dh^{l_{k+1}}_d}{dh^{l_k}_d}, \tag{3.17}$$

where $\frac{dh^{l_{k+1}}_d}{dh^{l_k}_d}$ $\forall k \in \{0, ..., n-1\}$ is expressed as:

$$\frac{dh^{l_{k+1}}_d}{dh^{l_k}_d} = \frac{d\phi(A^{l_{k+1}}_d)}{dA^{l_{k+1}}_d} \frac{dA^{l_{k+1}}_d}{dh^{l_k}_d} = \frac{d\phi(A^{l_{k+1}}_d)}{dA^{l_{k+1}}_d} W^{l_{k+1}}_{dd}. \tag{3.18}$$

$A^{l_k}_d$ denotes the pre-activation of $h^{l_k}_d$. The invertibility is satisfied by choosing a strictly increasing activation function ϕ (e.g. *tanh* or *sigmoid*) and a strictly positive

$W_{dd}^{l_k}$. $W_{dd}^{l_k}$ is made strictly positive by applying an element-wise exponential to all entries in $\forall d, k : W_{dd}^{l_k}$ at the end of the hypernetwork, inspired by [86]. Note that the operation is omitted for the non-diagonal elements of $W_{ij}^{l_k}, i \neq j$.

3.6.3 Tractable Computation of the Jacobian Determinant

The second requirement for Flow models is to efficiently compute the jacobian matrix determinant $\left|det\frac{dZ}{dX}\right|$, where:

$$\frac{dZ}{dX} = \frac{dZ}{dh^{l_n}} \prod_{k=0}^{n-1} \frac{dh^{l_{k+1}}}{dh^{l_k}} = W^{l_{n+1}} \prod_{k=0}^{n-1} \frac{d\phi(A^{l_{k+1}})}{dA^{l_{k+1}}} W^{l_{k+1}}. \tag{3.19}$$

Since we designed $W^{l_{k+1}}$ to be lower-triangular, the product of lower-triangular matrices, $\frac{dZ}{dX}$, is also lower-triangular, whose log determinant is then simply the product of the diagonal entries: $log\left|det\frac{dZ}{dX}\right| = log\left|\prod_{d=1}^{D} \frac{dz_d}{dx_d}\right| = \sum_{d=1}^{D} log(\frac{dz_d}{dx_d})$, as our formulation states $\forall d : \frac{dz_d}{dx_d} > 0$. Finally, $log(\frac{dz_d}{dx_d})$ is expressed via Equations 3.17 and 3.18.

$$log\left(\frac{dz_d}{dx_d}\right) = log\left(W_{dd}^{l_{n+1}} \prod_{k=0}^{n-1} \frac{d\phi(A_d^{l_{k+1}})}{dA_d^{l_{k+1}}} W_{dd}^{l_{k+1}}\right). \tag{3.20}$$

Equation 3.20 involves the multiplication of matrices of different sizes; thus cannot be broken down to a regular log summation. To resolve this issue, we utilize log-sum-exp operation on logs of the matrices in Equation 3.20 as it is commonly utilized in the Flow community (e.g. *NAF* [40] and *BNAF* [86]) for numerical stability and efficiency of the computation. This approach to computing the Jacobian determinant is similar to the one presented in *BNAF*, as our conditional AF resembles its Flow model.

As HCNAF is a member of the monotonic neural-net based autoregressive Flow family, like *NAF* and *BNAF*, we rely on the proofs presented in *NAF* and *BNAF* to claim that HCNAF is also a universal distribution approximator.

3.6.4 Hyper-conditioning and Training

The key point from Equation 3.12 - 3.20 and Figure 3.7 is that HCNAF is constraint-free when it comes to the design of the hyper-network. The Flow requirements from Sections 3.6.2 and 3.6.3 do not apply to the hyper-network. This enables the hyper-network to grow arbitrarily large and thus scale up to the dimension of conditions. The hyper-network $f_H(C)$ can therefore be an arbitrarily complex neural network with respect to the conditions C.

We seek to learn the target distribution $p(X|C)$ using HCNAF by minimizing the negative log-likelihood (NLL) of $p_{model}(X|C)$, i.e. the cross-entropy between the two distributions, as in:

$$L := -E_{X \sim p(X|C)}[log p_{model}(X|C)] = H(p, p_{model}). \tag{3.21}$$

Note that minimizing the NLL is equivalent to minimizing the (forward) KL divergence between the data and the model distributions $D_{KL}(p(X|C)||p_{model}(X|C))$, as $H(p,q) = H(p) + D_{KL}(p||q)$ where $H(p)$ is bounded.

Equation 3.21 can be further expanded the flow transformation of HCNAF (via Equation 3.9). As a result, the maximum likelihood estimation solution $p_{\pi_{MLE}}(X|C)$ is obtained as follows:

$$\begin{aligned}\pi_{MLE} &= \underset{\pi}{\text{argmax}} \mathbb{E}_X \big[\log p_\pi(X|C) \big] \\ &= \underset{\theta}{\text{argmax}} \mathbb{E}_X \left[\log N_{pdf}(f(X;\theta); 0, I) + \log \left| \det \frac{df(X;\theta)}{dX} \right| \right].\end{aligned} \tag{3.22}$$

3.6.5 HCNAF for Generative Tasks

The generative task denotes the problem of generating (i.e., predicting) the target variable X at the testing time. As HCNAF models the bijective transformation between $X \sim p(X|C)$ and $Z \sim N(0, I)$ and computes the exact value of $p(X|C)$, the generative task can be done in two different ways:

- Inverse operation of the flow: by leveraging the invertibility of the flow transformation, a sequence sample X is obtained using $X = f^{-1}(Z;\theta)$ where $Z \sim N(0, I)$.

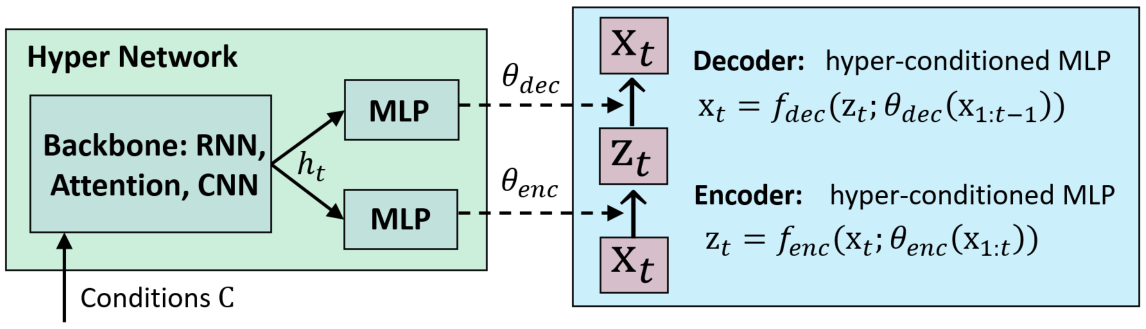
- Sampling via density estimation: using that normalizing-flow is capable of computing the exact probability, one can obtain the probability density of target variable X via Equation 3.9. The idea is to evaluate the probabilities of all possible realizations of X to formulate a list of probabilities $Y := [p(X_1|C), ..., p(X_N|C)]$ and sample from the list to obtain $X \sim Y$.

3.7 CVAE-H, A CVAE Augmented with Hypernetwork

Our second prediction model is CVAE-H (Conditional Variational AutoEncoder via Hypernetwork) [6], a conditional variational autoencoder that integrates a hypernetwork into a VAE. Similar to HCNAF, the hypernetwork encodes various conditioning information (e.g., social and spatial information) and the VAE utilizes the encoding as the network parameters and auto-encodes the target variables.

While VAE computes the probability density approximately using the lower bound, backbone neural networks of VAE are less constrained compared to those of normalizing flow since VAE does not enforce the invertibility constraints to the backbone networks as Flow does. In practice, VAE-based models have demonstrated competitive results for trajectory prediction tasks [30, 54, 42] as well as other machine learning tasks such as sequence modeling and density estimation.

Hypernetwork conditionalizes the vanilla (unconditional) VAE in a similar fashion to how hypernetwork transformed NAF into HCNAF (see Section 3.6). As elaborated in Section 3.5.3, one advantage of hypernetwork over the embedding approach is the flexibility of hypernetwork (i.e., any differentiable neural network can be used as a hypernetwork). Another advantage is that hyper-network can encode conditioning information with a smaller number of network parameters compared to the embedding approach [81].



Hyper Network
Backbone: RNN, Attention, CNN
h_t
MLP
MLP
Conditions C
θ_{dec}
θ_{enc}
x_t
z_t
x_t
Decoder: hyper-conditioned MLP
$x_t = f_{dec}(z_t; \theta_{dec}(x_{1:t-1}))$
Encoder: hyper-conditioned MLP
$z_t = f_{enc}(x_t; \theta_{enc}(x_{1:t}))$

Figure 3.8: Schematic of CVAE-H. The inputs of CVAE-H consist of conditions C and a target variable x (for training) or sample latent variable z (for inference). For the prediction problem, we use the following conditions $C := [S_{0:t}^{A_{\forall k}}, \Omega]$ and target variable $X_t := [x_t^{AV}, y_t^{AV}]$.

Figure 3.8 illustrates CVAE-H, which largely consists of two modules: 1) a VAE and 2) a hypernetwork that computes the network parameters of 1). In the following subsections, the mathematical details of the two modules are presented.

3.7.1 Hyper-conditioning VAE

Conditioning VAE via a hypernetwork involves the following process of conditioning the encoder and decoder:

$$
\begin{aligned}
&(Encoder) \quad Z = f_{enc}(X; \theta_{enc}(C)) \\
&(Decoder) \quad X = f_{dec}(Z; \theta_{dec}(C))
\end{aligned}
\tag{3.23}
$$

where θ_{enc} and θ_{dec} are obtained via a hypernetwork $f_H(C)$, that is, $f_H(C) = [\theta_{enc}, \theta_{enc}]$. If a VAE is conditionalized using the embedding approach, the resulting equations would instead be $Z = f_{enc}(X, g_1(C))$ and $X = f_{dec}(Z, g_2(C))$. For the reasons described in Section 3.5.3, we conditionalized the VAE using a hypernetwork.

The VAE network of CVAE-H can consist of any type of neural network. For a 1D target variable such as a sequence or a collection of variables, MLP is a good choice. While we expect CNN to be an effective encoder & decoder structure for a 2D target variable such as an image, we leave the search of proper network for 2D target variables to the future work because it is out of the scope of this work.

Similar to how $p(X)$ is modeled in an unconditional VAE, a latent variable Z is introduced to map the inputs X and condition C to the the conditional distribution $p(X|C)$ as follows: $p(X|C) = \int_Z p(X|Z,C)p(Z|C)dZ$. Here, $p(Z|C)$ and $p(X|Z,C)$ represent the prior and the generative decoder conditioned on C, respectively. Ideally, Z should encode abstract features of $p(X|C)$.

Since the integral is intractable, variational inference [59] is used to approximate the integral with a parameterized posterior $q_{\theta_{enc}}(Z|X,C)$. This leads to the construction of ELBO on the log conditional likelihood as follows:

$$\log p(X|C) \geq \mathbb{E}_{q_{\theta_{enc}}(Z|X,C)}[\log p_{\theta_{dec}}(X|Z,C)] - \mathrm{KL}(q_{\theta_{enc}}(Z|X,C)||p(Z|C)). \quad (3.24)$$

We set the prior to be a uni-modal Gaussian with the diagonal covariance whose entries are equal to one. Different choices of the prior include parameterizing priors using neural networks and/or domain knowledge [88].

3.7.2 Hypernetwork and Training of CVAE-H

The hypernetwork aims to effectively propagate the conditioning information to VAE. This is achieved by customizing the backbone neural networks of the hypernetwork. For sequence models and/or language modelling, a hypernetwork can include RNN, attention modules, and/or Transformer [71]. For computer vision tasks, a hypernetwork may consist of CNN-based models. The use of hypernetwork for CVAE-H is very similar to that for HCNAF (see Section 3.6), except the hypernetwork for CVAE-H outputs two sets of hyper-parameters $[\theta_{enc}, \theta_{dec}]$ each for the encoder and decoder of VAE, whereas the hypernetwork for HCNAF outputs only θ.

The goal of CVAE-H is to learn the target conditional distribution $p(X|C)$ by minimizing the negative log-likelihood of model distribution $p_{model}(X|C)$. This is achieved by maximizing the ELBO presented in Equation 3.24. The first term of the ELBO is the reconstruction error of conditional VAE and can be computed as long as the estimation of $p_{\theta_{dec}}(X|Z, C)$ is tractable. For this reason, we design the encoder to be an explicit density model (e.g., softmax, MDN). The second term of the ELBO is a KL divergence and can be easily computed as we use Gaussian distributions for both posterior $q_{\theta_{enc}(Z|X,C)}$ and prior $p(Z|C)$.

3.7.3 CVAE-H for Generative Tasks

The training and inference processes for CVAE-H are different. During the training, we have access to the target variable X. By passing X to the encoder $Z = f_{enc}(X; \theta_{enc}(C))$, we obtain the posterior Z. The decoder then takes Z to recover $X = f_{dec}(Z; \theta_{dec}(C))$. During the inference, the encoder is not used because we do not have access to X. Instead, we leverage the known prior distribution $p(Z|C)$, sample $Z \sim p(Z|C)$, and pass C to the hyper-network to compute θ_{dec}, which constructs f_{dec}. Lastly, we pass θ_{dec} and Z through the decoder to obtain $X = f_{dec}(Z; \theta_{dec}(C))$. In this sense, we only run the decoder and part of the hypernetwork, which outputs θ_{dec}, for the generation task.

3.8 Forecasting with HCNAF and CVAE-H

In the previous two sections, we presented the fundamentals of HCNAF and CVAE-H. This section explains how we leverage their attributes to tackle the prediction task for autonomous driving.

A challenge of the prediction task is that it may involve very large inputs whose dimensions are in the magnitude of millions. This is because the inputs include social features (e.g., labeled states for all road-agents in the vicinity of the AV) and spatial features (e.g., lidar scans, camera images, and/or HD maps) from past to the current moment. Suppose the inputs consist of a sequence of 128 by 128 rasterized images with 6 channels, or lidar tensors of the same dimension, over 10 past time steps. In that case, the dimensionality is approximately one million, as in $C \in \mathbb{R}^{T \cdot C \cdot H \cdot W \approx 1{,}000{,}000}$.

Both HCNAF and CVAE-H have *high modeling capacity* and allow the AV to operate on the high-dimensional inputs and produce trajectory or POM forecasting contextual to the large inputs. This is possible because the hypernetworks of HCNAF and CVAE-H take the inputs as conditions C, effectively encoding them as θ that captures essential information about the environment. Depending on the architecture of the hyper-network's backbone networks, HCNAF and CVAE-H can be customized to different machine learning tasks. In this regard, the backbone of the hyper-network for autonomous driving should be designed to best exploit the information that comes from various sensors of self-driving vehicles.

3.8.1 Hypernetwork Design

Considering the state of environment $S_t := [X_t^{A_{\forall k}}, V_t^{A_{\forall k}}, \Omega_t]$ defined in Equation 2.1, we design the hypernetwork to include (1) a social module, (2) a spatial module, and (3) a time module.

The social module $f_H^S(X_{0:t}^{A_{\forall k}}, V_{0:t}^{A_{\forall k}}) = h^S$ aims to capture social features among the road-agents. The module takes past states of road agents including positions and speeds $X_{0:t}^{A_{\forall k}} \in \mathbb{R}^{t \times N_A \times 2}$, $V_{0:t}^{A_{\forall k}} \in \mathbb{R}^{t \times N_A \times 2}$ where N_A denotes the number of road-agents and outputs abstract social hidden states h^S. Sequence models such as RNN, attention, Transformers, and graph neural networks (GNN) are possible candidates to the social module. In this work, we use RNN and attention as the social module.

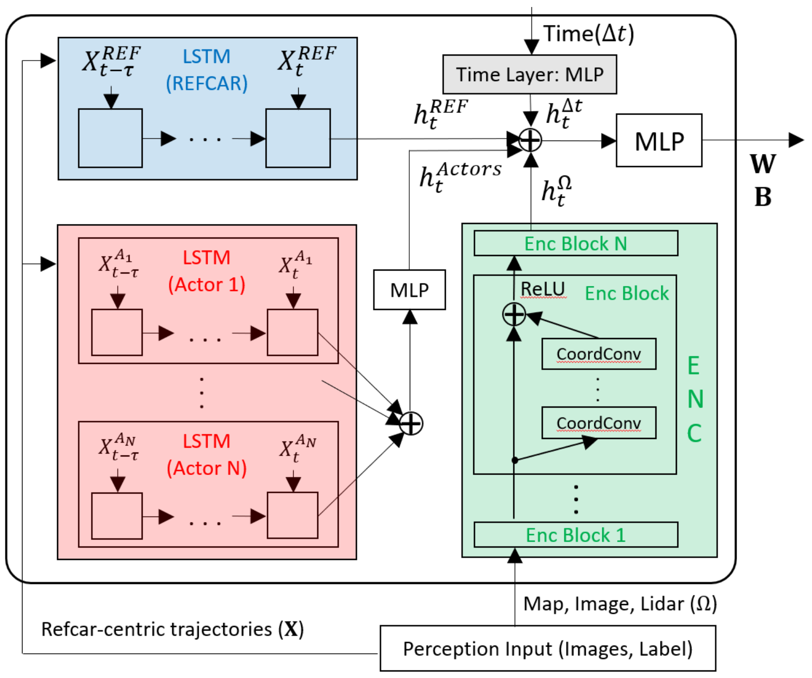


Figure 3.9: The hypernetworks of HCNAF and CVAE-H customized to the prediction task

The spatial module $f_H^{\Omega}(\Omega_t) = h^{\Omega}$ aims to capture the spatial contexts into the prediction. This module takes spatial data $\Omega \in \mathbb{R}^{N_C \times H \times W}$ (N_C, H, W that represents the number of channels, height, width of the bird's-eye-view (BEV) map) such as lidar point clouds, camera images, and/or HD maps represented as rasterized BEV tensors with N_C channels. The outputs of the spatial module are abstract spatial hidden states h^{Ω}. We leverage variants of CNN-based models such as ResNet [94] along with max-pooling layers, and batch-normalization layers to extract the spatial features from Ω. We also utilize coordinate convolution (coordconv) layers [95] to strengthen the association between the two different types of inputs; image (i.e., pixel) data $\Omega \in \mathbb{R}^{N_C \times H \times W}$ and Cartesian coordinate-based sequence data $X_{0:t}^{A_{\forall k}} \in \mathbb{R}^{t \times N_A \times 2}$, $V_{0:t}^{A_{\forall k}} \in \mathbb{R}^{t \times N_A \times 2}$ used in the social module.

The time module $f_H^T(\delta t) = h^T$ is a relatively simple module that takes the forecasting time $\Delta t \in \mathbb{R}^1$ (i.e. time span of the future t away from the reference time t) and outputs abstract time hidden states h^T. The module aims to control the temporal trend of the prediction outputs. We apply an MLP for the time module.

After we obtain the abstract features h^S, h^Ω, h^T, they are concatenated and passed through an MLP to obtain the network parameters of the main network θ, which is the end product of the hypernetwork.

Figure 3.9 illustrates an example design of the hypernetwork. It is worth noting that we design the backbone networks for HCNAF and CVAE-H identically. This is to set the conditioning capability as similar as possible for fair comparisons of the two models. This is possible because the architectural design of the hypernetworks of HCNAF and CVAE-H are very similar, as portrayed in Figure 3.6 and 3.8.

3.8.2 Forecasting Trajectories

In this subsection, we describe how we utilize our prediction models to generate trajectory predictions, which is one of the two prediction representations (see Section 2.3). As defined in Section 2.1, we denote $X^{A_k}_{t:T} := [x^{A_k}_{t:T}, y^{A_k}_{t:T}]$ as the trajectory of the road-agent A_k from t to T.

We obtain trajectory samples by querying the conditional probability $p(X^{A_k}_{t+1:T}|C)$ that are modeled using HCNAF and CVAE-H as follows:

$$\text{(Trajectory Forecasting)} \quad X^{A_k}_{t+1:T} \sim p(X^{A_k}_{t+1:T}|C),\ \forall k \neq AV, \tag{3.25}$$

where $C := S_{0:t}$ is the environment states defined in Equation 2.1. However comprehensive, the list of conditions in C is not complete; as additional cues are introduced to better define road-agents or enhance context, those can be appended to the conditions.

While HCNAF and CVAE-H are able to model $p(x^{A_{\forall k}}_{t:T}, y^{A_{\forall k}}_{t:T}|C)$ (i.e., the joint distribution of all road-agents' trajectories) at once, we instead model them to only compute $p(x^{A_k}_{t:T}, y^{A_k}_{t:T}|C)$ (i.e. the distribution of a single road-agent A_k) for every road-agent except the AV (see *non-autoregressive* approach discussed in Section 2.2 for details). This way, the prediction models can address varying number of road-agents. The compounding error problem is another factor for the consideration; the computation of $p(x^{A_{\forall k}}_{t:T}, y^{A_{\forall k}}_{t:T}|C)$ implies the autoregressive formulation $p(x^{A_{\forall k}}_{i}, y^{A_{\forall k}}_{i}|x^{A_{\forall k}}_{1:i-1}, y^{A_{\forall k}}_{1:i-1}, C)$ over $i = t : T$. This formulation reasons about the temporal dependencies of the social features, however, it forces the models to make predictions on $x^{A_{\forall k}}_{i}, y^{A_{\forall k}}_{i}$ dependent on unobserved variables $x^{A_{\forall k}}_{t:i-1}$ and $y^{A_{\forall k}}_{t:i-1}$. The uncertainties of the unobserved variables have the potential to push the forecast $x^{A_{\forall k}}_{t:T}, y^{A_{\forall k}}_{t:T}$ in the wrong direction; hence, the compounding error problem arises.

The trajectory prediction task is essentially the generative task described in Section 3.6.5 and 3.7.3. For this reason, we refer to those sections for the methodological details.

3.8.3 Forecasting POM

POM is the second prediction representation (see Section 2.3) and can be obtained using HCNAF or CVAE-H. The following equation describes how POM $O \in \mathbb{R}^{T\times H\times W}$ is defined.

$$\text{(POM Forecasting)} \quad O_i(X_i) := p(X_i|C), \ t < i \leq T, \tag{3.26}$$

where $O_t \in \mathbb{R}^{H\times W}$ denotes the POM at time t and $O_t(x, y) = p(x_t, y_t|C) \in \mathbb{R}^1$ represents the probability occupancy at the coordinate (x, y).

Recall that given a coordinate $X_t = (x_t, y_t)$, our prediction models can estimate the probability occupancy $p(X_t|C)$ either exactly (HCNAF) or approximately via ELBO (CVAE-H). For discrete POM, we compute $p(X_t|C)$ for all $X_t \in S_{X_t}$ to obtain O_t, where S_{X_t} represents a set of target coordinates at t. It is worth noting that in the experiments elaborated later in this Chapter, we only use HCNAF to obtain POM since CVAE-H only approximates the lower bounds of the probability occupancy that are difficult to interpret.

It should be noted that O_{t_1} is conditionally independent of O_{t_2} for $\{(t_1, t_2)|t_1 \neq t_2, t < t_1, t_2 \leq T\}$ given C. In other words, we model $p(X_t|C)$ independently at each time instance (marginal distribution over a single time instance), instead of modelling the joint distribution $p(X_{t:T}|C)$ over all time instances for the following reasons:

1. Computing $p(X_{t:T}|C)$ implies an autoregressive factorization $p(X_i|X_{t:i-1}, C)$ over $t < i \leq T$. While this formulation reasons about the temporal dependencies between the history and the future, it forces the model to make predictions on X_t dependently on unobserved variables X_{t-1}. The uncertainties of the unobserved variables have the potential to push the forecast X_t in the wrong direction (i.e., the compounding error problem).

2. The decision-making task requires the access to the marginal distribution $p(X_t|C)$. Suppose we model $p(X_{t:T}|C)$. Then we need to marginalize the joint distribution over all other variables X_t from t to T. The marginalization $p(X_t|C) = \int_{-\infty}^{\infty} ... \int_{-\infty}^{\infty} p(X_t, ..., X_{T-dt}) dX_t ... dX_{T-dt}$ is practically impossible (i.e., intractable).

In order to adapt HCNAF and CVAE-H to work with $X_t \in \mathbb{R}^2$, we obtain the joint probability $p(x_t, y_t|C) = p(y_t|x_t, C)p(x_t|C)$.

3.9 Evaluation of the Models: Toy Experiments

Before evaluating HCNAF and CVAE-H for the self-driving prediction task, we first assess them in density estimation and generation tasks often used to evaluate in the literature [39, 40, 87, 90, 96]. The task of forecasting POM is essentially a density estimation task. In this sense, density estimation tasks allow us to evaluate how well HCNAF learns and estimates probability densities of target distributions. Likewise, the task of forecasting a trajectory is fundamentally a generative task. Therefore, generative tasks help us evaluate the quality of samples generated from CVAE-H.

More importantly, the experiments examine the capacity of HCNAF and CVAE-H as an ideal prediction model introduced in Section 3.1. In other words, we validate if the models are probabilistic, multi-modal, context-driven, and general using a number of toy experiments of different complexities. The toy experiments allow us to evaluate HCNAF and CVAE-H in simpler and more interpretable settings than the self-driving prediction task. They provide us with better insights regarding the strengths and weaknesses of HCNAF and CVAE-H.

In this section, two density estimation experiments, namely *Gaussian 1* and *Gaussian 2* are presented to verify whether HCNAF and CVAE-H have the four attributes of the ideal prediction model. We train HCNAF for density estimation and CVAE-H for sample generation. To evaluate the performance of the models, we utilize both quantitative and qualitative measures. The quantitative measures include NLL and KL divergence (D_{KL}), which are defined in the following.

$$\begin{aligned} NLL &= -E_{X\sim p(X|C)}[\log(p_{model}(X|C))], \\ D_{KL}(p||p_{model}) &= \sum_{X\sim p(X|C)} p(X|C)\log\left(\frac{p(X|C)}{p_{model}(X|C)}\right). \end{aligned} \tag{3.27}$$

To evaluate HCNAF and CVAE-H qualitatively, we visualize the outputs of the models. Recall that HCNAF outputs probability densities $p(X|C)$ and CVAE-H outputs sample coordinates $X \sim p(X|C)$. In this regard, the visualizations include the resulting probability densities from HCNAF and generated samples from CVAE-H.

Figure 3.10 portrays the HCNAF and CVAE-H network designs customized for the Gaussian experiments. Note that both HCNAF and CVAE-H leverage hypernetworks to extract the conditioning information.

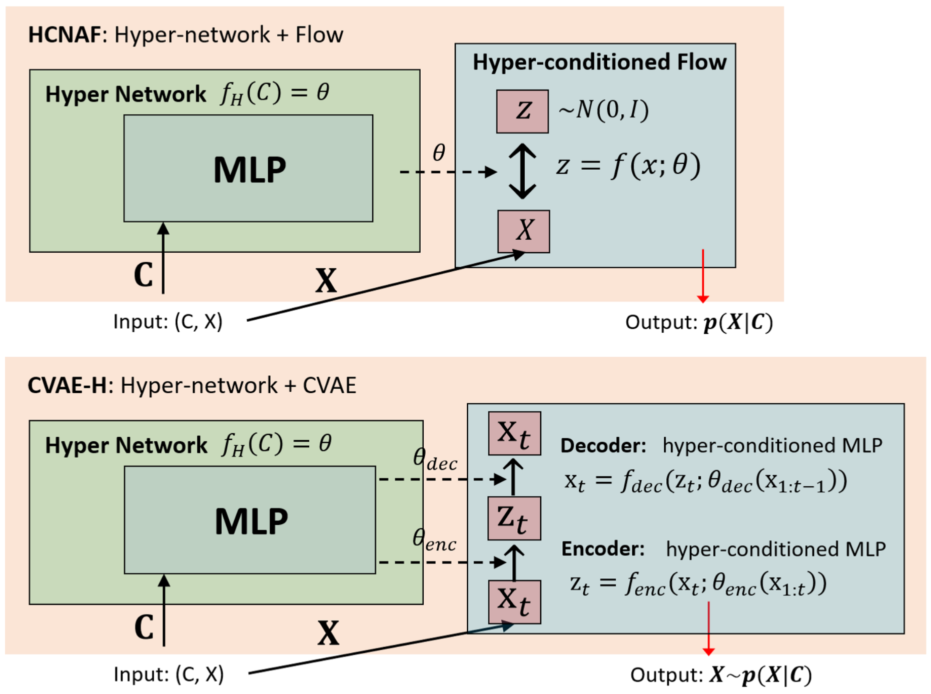

HCNAF: Hyper-network + Flow
Hyper Network $f_H(C) = \theta$
MLP
θ
Hyper-conditioned Flow
Z $\sim N(0, I)$
$z = f(x; \theta)$
X
C
X
Input: (C, X)
Output: $p(X|C)$
CVAE-H: Hyper-network + CVAE
Hyper Network $f_H(C) = \theta$
MLP
θ_{dec}
θ_{enc}
X_t
Z_t
X_t
Decoder: hyper-conditioned MLP
$x_t = f_{dec}(z_t; \theta_{dec}(x_{1:t-1}))$
Encoder: hyper-conditioned MLP
$z_t = f_{enc}(x_t; \theta_{enc}(x_{1:t}))$
C
X
Input: (C, X)
Output: $X \sim p(X|C)$

Figure 3.10: HCNAF and CVAE-H network designs customized for the Gaussian experiments. The inputs to the HCNAF and CVAE-H are a tuple of target variable X and conditioning information C where X corresponds to the (x, y) coordinates and C corresponds to a set of discrete class variable $C \in \{0, 1, 2\}$ for the Gaussian 1 experiment and $C \in \{(0, 0), (-4, 4), (-4, 4), (4, -4), (4, 4)\}$ for the Gaussian 2 experiment.

3.9.1 The Gaussian 1 Experiment

The Gaussian 1 experiment is the experiment used in [40] and aims to show the model's learning ability for three distinct multi-modal probability distributions $p_1(x, y), p_2(x, y), p_3(x, y)$ of 2-by-2, 5-by-5, and 10-by-10 Gaussians depicted in Figure 3.11. We compare the results against the recent state-of-the-art density estimator NAF [40] as well as affine autoregressive flow (AAF). NAF and AAF were both used as density estimators like HCNAF.

We used the same number of hidden layers (2), hidden units per hidden layer (64), and batch size (64) across all models, including AAF, NAF, and HCNAF. For NAF, we utilized the conditioner (transformer) with 1 hidden layer and 16 sigmoid units, as suggested in [40]. For HCNAF, we modeled the hyper-network with two MLPs, each taking a condition $C \in \mathbb{R}^1$ and outputs θ. Each MLP consists of 1 hidden layer, a ReLU activation function. For CVAE-H, we used the identical hyper-network as HCNAF. The encoder and decoder of CVAE-H are both designed using 4-layer MLPs. The last layer of the decoder outputs parameters (means, variances, and mixing probabilities) of a Gaussian mixture. The outputs of CVAE-H are generated by sampling the output Gaussian mixture. All the other parameters were set identically, including those for the Adam optimizer (the learning rate $5e^{-3}$ decays by a factor of 0.5 every 2,000 iterations with no improvement in validation samples). The NLL (i.e., $p_{model}(X_{target})$) values in Table 3.1 were computed using 10,000 samples.

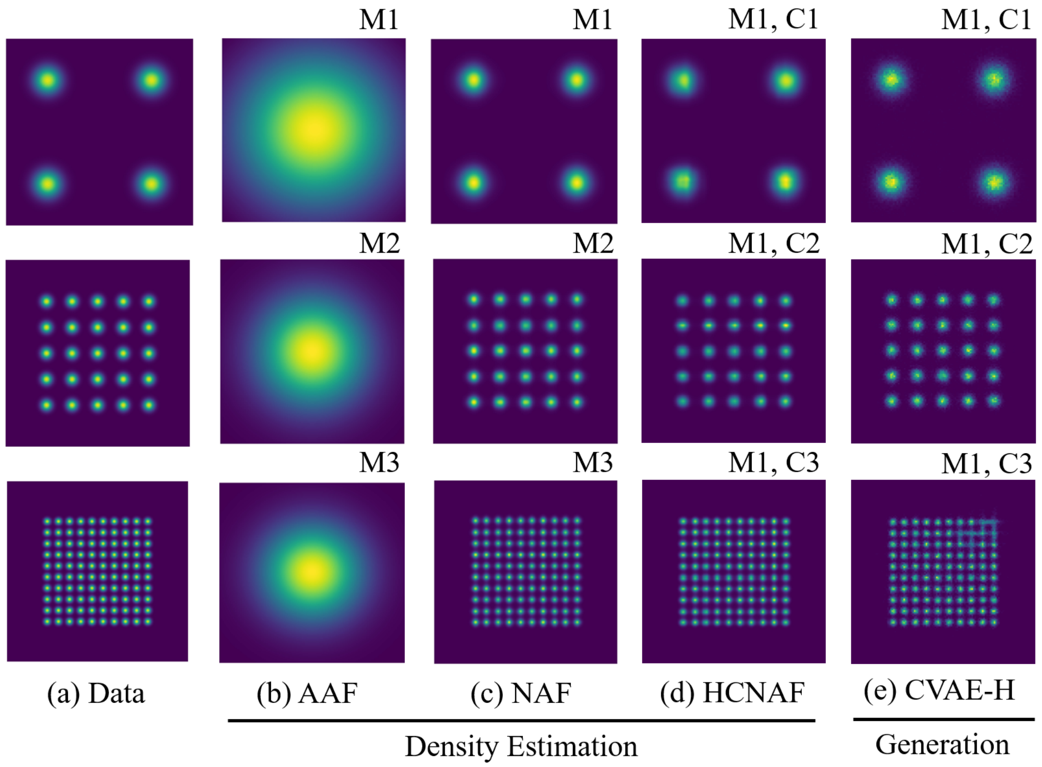


Figure 3.11: Qualitative result of the Gaussian 1 experiment. AAF, NAF, HCNAF were tested for density estimation and CVAE-H was tested for sample generation. M and C each denotes model and condition respectably.

In order to reproduce the three distributions, AAF and NAF require three different and separately trained models $p_1(x, y; \theta_1), p_2(x, y; \theta_2), p_3(x, y; \theta_3)$. Conversely, HCNAF and CVAE-H uses a *single* model and three conditions. In other words, HCNAF and CVAE-H model $p(x, y|C_k; \theta), k = 1, 2, 3$ using one set of parameters θ. The conditioning information $C \in \{0, 1, 2\}$ where each value represents the class of the 2-by-2, 5-by-5, and 10-by-10 Gaussians.

Table 3.1: NLL for the experiment depicted in Figure 3.11. Lower values are better.

	AAF	NAF	HCNAF (ours)	CVAE-H (ours)
2 by 2	6.056	3.775	3.896	3.895
5 by 5	5.289	3.865	3.966	3.978
10 by 10	5.087	4.176	4.278	4.292

Results from Figure 3.11 and Table 3.1 show that HCNAF and CVAE-H are both able to reproduce the three nonlinear target distributions, confirming that they are *probabilistic* and *multi-modal*. They also achieve comparable results as NAF, albeit with a small increase in NLL. It is important to note that our models use a *single* model (with a 1-dimensional condition variable) to produce the 3 distinct pdfs, whereas AAF and NAF used *3 distinctly trained* models. One HCNAF or CVAE-H model $p(x, y|C_k; \theta), k = 1, 2, 3$ is able to accurately reproduce very different nonlinear conditional target distributions. This confirms that HCNAF and CVAE-H are context-driven.

We point out that the plots displayed in Figure 3.11 contain results from both density estimation (for AAF, NAF, and HCNAF) and sample generation (for CVAE-H). In the density estimation task, we estimate $p_{model}(x, y|C_k)$ where (x, y) belongs to a cell in a discrete 2-dimensional grid. In other words, $(x, y) \in \text{Grid}([x_{min}, y_{min}], [x_{max}, y_{max}])$. In the sample generation task, we first obtain $(x, y) \sim p_{model}(x, y|Z, C_k; \theta), k = 1, 2, 3$ where Z is sampled from the prior. Then, we use a histogram to classify the generated (x, y) into the 2-dimensional grid. While we report NLLs for HCNAF and CVAE-H in the same table, we do not make a direct comparison of their NLLs since the density estimation and generative task are two different tasks. Precisely, the NLL for HCNAF was estimated by evaluating $p_{model}(X_{target}|C_k)$ whereas the NLL for CVAE-H was obtained by computing $p_{model}(X_{target}|Z, C_k), Z \sim N(0, I)$.

3.9.2 The Gaussian 2 Experiment

While the Gaussian 1 experiment is about reproducing the target distributions that the models were trained on, the Gaussian 2 experiment is designed to evaluate how HCNAF and CVAE-H can generalize their outputs over *unseen* inputs. By assessing the models with a set of conditions beyond what it was trained with, we validate the models' capacity to interpolate and extrapolate.

Similar to the Gaussian 1 experiment, we train a single HCNAF model to learn five distinct pdfs, where each pdf represents a Gaussian distribution with its mean used as conditions $C := (x_c, y_c) \in \mathbb{R}^2$ and an isotropic standard deviation σ of 0.5.

We train the models with five different discrete conditions $C_{train} = \{C_1, ..., C_5\}$, where C_i represents the mean of an isotropic bivariate Gaussian pdf. We then check how accurately the models (1) reproduce the data distribution $p(x, y|C_{train})$ it was trained on and (2) predict new distributions $p_{model}(x, y|C_{unseen})$, $C_{unseen} := \{C_6, ..., C_9\}$ that the models have not seen before.

We used 3 hidden layers, 200 hidden units per hidden layer for the hyper-conditioned flow. We use the same hypernetwork architecture in Gaussian 2 experiment as the Gaussian 1 experiment. We trained the model with a batch size of 4. The NLL values in Table 3.2 and 3.3 were computed using 10,000 test samples from the target conditional distributions.

The qualitative results presented in Figure 3.12 suggest that HCNAF is capable of *generalizing* over *unseen* conditions, i.e. values in the condition terms that were intentionally omitted during training.

Table 3.2 provides quantitative results from the cross-entropy $H(p, p_{model})$ and the KL divergence $D_{KL}(p||p_{model})$. Note that $H(p, p_{model})$ is lower-bounded by $H(p)$ since $H(p, p_{model}) = H(p) + D_{KL}(p||p_{model})$. The differential entropy $H(p)$ of an isotropic

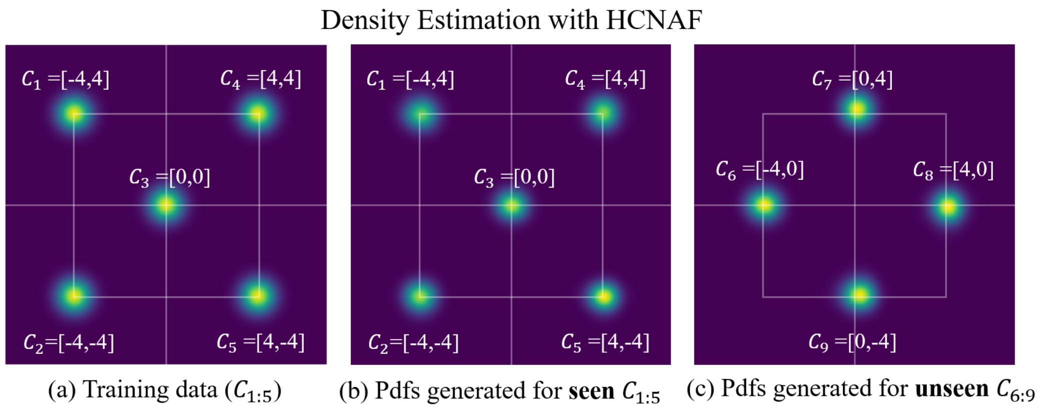


Figure 3.12: Qualitative results of the Gaussian 2 experiment for density estimation with HCNAF. (a) data distribution $p(x, y|C_{train})$, b) the data distribution reproduced by HCNAF $p_{model}(x, y|C_{train})$, and c) HCNAF's density estimation on previously *unseen* conditions $p_{model}(x, y|C_{unseen})$.

Table 3.2: Differences between the target entropy and recreated pdfs by HCNAF in terms of cross-entropy and KL divergence for Figure 3.12.

	$\boldsymbol{p(x, y)}$	$\boldsymbol{p_{HCNAF}(x, y\|C_i)}$	
C	-	$C_i \in C_{train}$	$C_i \in C_{unseen}$
$H(p)$	1.452	-	-
$H(p, p_{model})$	-	1.489	1.552
$D_{KL}(p\|\|p_{model})$	-	0.037	0.100

bi-variate Gaussian distribution $p(x, y)$ is computed using: $H(p) = 0.5 \cdot ln(2\pi e(\sigma)^2)^2$. Recall that minimization of NLL (i.e., the learning objective) is equivalent to the minimization of the KL divergence $-E_{(x,y)\sim N(C_i, 0.25\cdot I)}[logp_{model}(x, y|C_i)]$ where C_i is uniformly sampled from the set of conditions $C_{train} := \{C_1, C_2, ..., C_5\}$.

On the other hand, we present the results of the generative task with CVAE-H qualitatively in Figure 3.13 and quantitatively in Table 3.3, similar to the results presented for HCNAF. The generative cross-entropy for seen conditions (i.e., $H(p, p_{model}(X_{target}|Z, C_{seen})))$ are very close to the target differential entropy, even compared to HCNAF's cross-entropy. However, the generative cross-entropy for unseen conditions is relatively higher. This quantification agrees with the generated pdfs illustrated in Figure 3.13. While CVAE-H is capable of generalizing over unseen

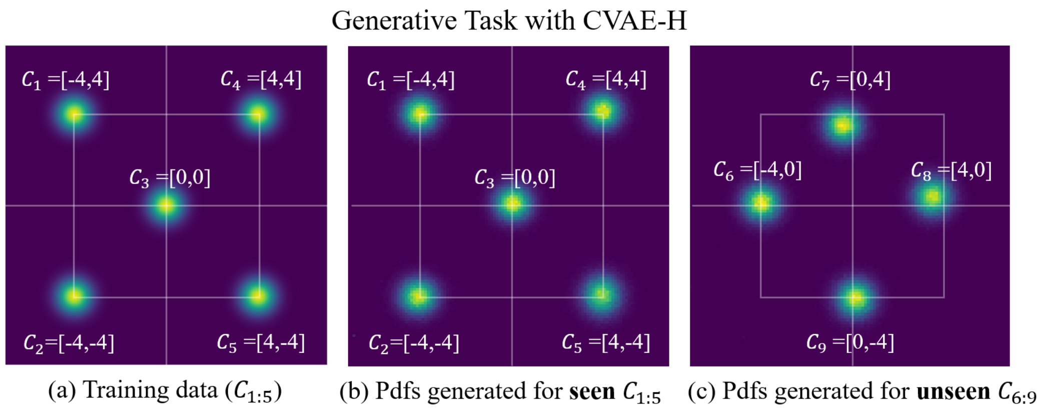


Figure 3.13: Qualitative results of the Gaussian 2 experiment for the generation task with CVAE-H. (a) data distribution $p(x, y|C_{train})$, b) the data distribution reproduced by CVAE-H $p_{model}(x, y|Z, C_{train})$, and c) CVAE-H's sample generation in *unseen* conditions $p_{model}(x, y|Z, C_{unseen})$.

conditions, HCNAF outperformed CVAE-H in terms of the quality of the generalization.

Table 3.3: Differences between the target and *generative* distributions by CVAE-H in terms of cross-entropy and KL divergence for Figure 3.13.

	$p(x,y)$	$p_{CVAE-H}(x,y\|Z,C_i)$	
C	-	$C_i \in C_{train}$	$C_i \in C_{unseen}$
$H(p)$	1.452	-	-
$H(p, p_{model})$	-	1.480	2.256
$D_{KL}(p\|\|p_{model})$	-	0.028	0.804

In summary, this section presented the results of the toy experiments and empirically verified that both HCNAF and CVAE-H satisfy all four requirements of the ideal prediction model; probabilistic, multi-modal, context-driven, and general.

3.10 Evaluation of the Models: Generic Urban Driving

In this section, we demonstrate the performance of HCNAF and CVAE-H models in urban driving scenarios using the PRECOG-Carla dataset.

3.10.1 Introduction to the Dataset

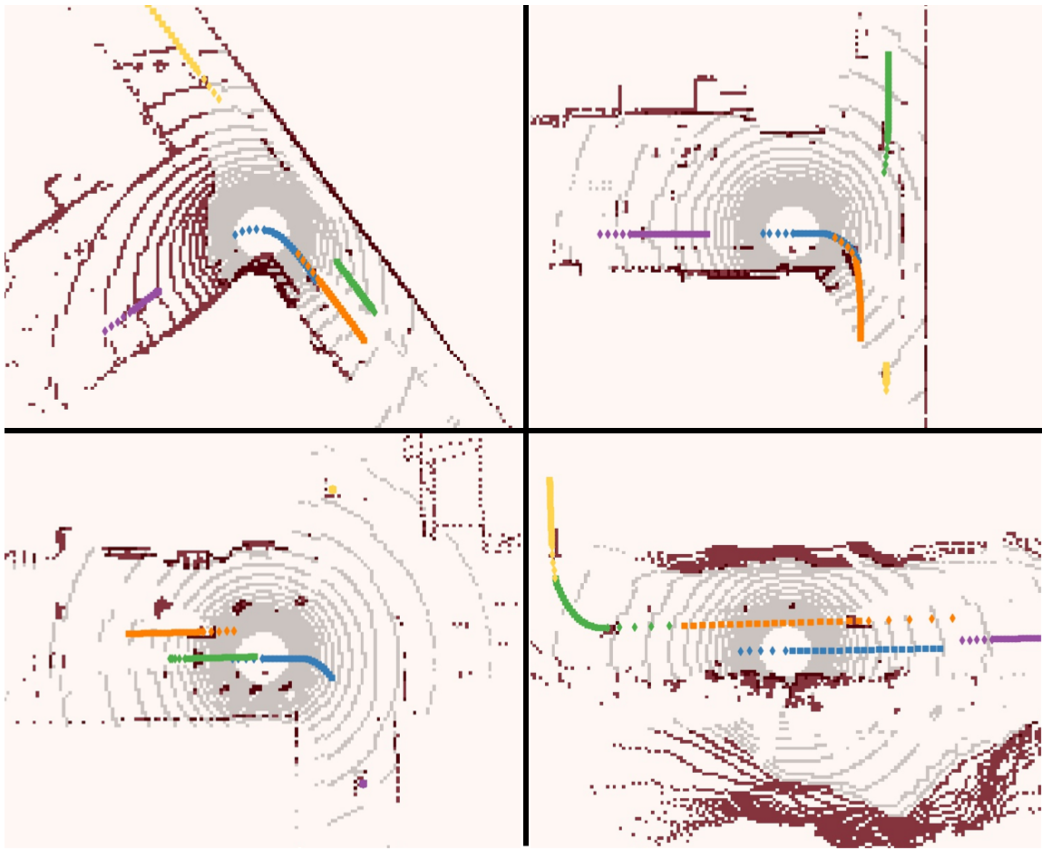


Figure 3.14: Four examples of the dataset. Each example includes lidar scans of an environment projected onto a 2D BEV map, the ego-vehicle depicted in blue, and up to 4 road-agents depicted in yellow, green, purple, and orange. Each trajectory is 6s long. We use the first 2s (diamond) as the inputs and forecast the last 4s (square) future positions.

As introduced in Section 2.4, PRECOG-Carla is a publicly available prediction dataset created using the open-source Carla simulator for autonomous driving research. PRECOG-Carla includes roughly 76,000 urban driving examples each consists of a 6-second scenario, with up to five road-agents (i.e., simulated human-driven vehicles) driving in urban areas and lidar point clouds of the environment. The lidar data provides three overhead lidar channels (i.e., two above ground and one ground level inputs), so the input to the spatial module is the raw lidar data $\Omega \in \mathbb{R}^{3\times200\times200}$.

Figure 3.14 describes 4 of the 76,000 cases of the dataset we work with. The ground-level lidar data is colored gray and the above-ground level channel is colored dark red. Each sample spans 6 seconds in total. The first two seconds (i.e., $t = -2 : 0s$) are depicted as diamonds and are used as inputs to predict the future 4-second (i.e., $t = 0 : 4s$) trajectories of the road-agents and AV. The ground-truth future trajectories are depicted as squares. Our goal is to produce predictions of the road-agents in the scene in the form of POM heat maps using HCNAF and trajectory samples using CVAE-H.

For the multi-agent forecasting with CVAE-H, we apply coordinate transformations to obtain *target-centric* coordinates as we found this helps to improve the accuracy of the multi-agent forecasting. Specifically, the coordinates of all give vehicles in the scene are transformed so that A_k, the target vehicle of the prediction, is positioned at (0,0) at zero heading angle at $t = 0$. That is, $X_{t=0}^{A_k} := [0, 0]$ and $V_{t=0}^{A_k} \cdot [0, 1] = 0$. The other road-agents' positions are transformed using the same transformation matrix. As there exists five road-agents in each example, we create 5 instances with each and every one of the road-agents to be located at the center $(0, 0)$ at $t = 0$. That is, $[X_{-2:4s}^{A_{\forall k}}]^{A_m}, m \in 1, 2, 3, 4, 5$, where $[X_{-2:4s}^{A_{\forall k}}]^{A_m}$ indicates that $X_{-2:4s}^{A_{\forall k}}$ is transformed in the A_m-centric coordinates.

For the POM forecasting, we did not perform a coordinate transform because the occupancy forecasting is not dependent on the road-agents. In this regard, we leverage the AV-centric coordinates for predicting POMs.

The 76,000 clips in the PRECOG-Carla Town01 dataset are split into a training set (80%), validation set (10%), and test set (10%). Both HCNAF and CVAE-H are trained on the PRECOG-Carla Town01-train dataset and the progress was validated over Town01-val dataset. Town01-test dataset is used to evaluate the performance of the models. It is important to note that *all qualitative and quantitative results presented in this thesis are based on the test set.* In other words, the models were tested with *unseen* data.

3.10.2 Hyper-network Design

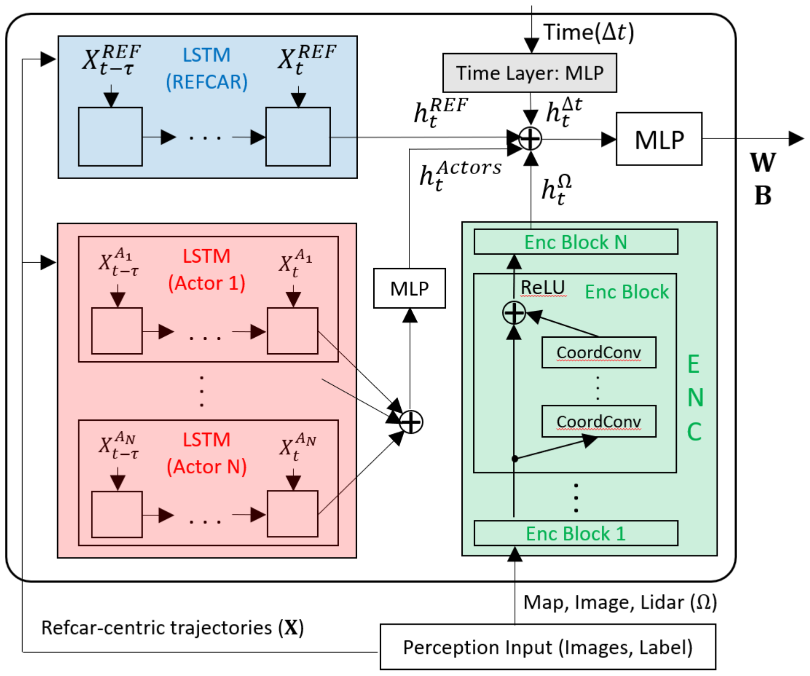


Figure 3.15: The hypernetwork design for the urban driving dataset.

Figure 3.15 depicts the hypernetwork design customized for the prediction problem. The hypernetwork takes perception inputs as the condition C, and outputs a set of network parameters θ. For HCNAF, θ is the network parameters of the subsequent hyper-conditioned flow $f(\cdot;\theta) : X \leftrightarrow Z \sim N(0, I_{2x2})$. For CVAE-H, $\theta = [\theta_{enc}, \theta_{dec}]$ includes network parameters for the encoder $Z = f_{enc}(X;\theta_{enc})$ and decoder $X = f_{dec}(Z;\theta_{dec})$.

Specifically, C is formed with (1) the spatial data on a BEV map and (2) the social data such as states of road-agents. As aforementioned, in POM forecasting, the states of the road-agents are in AV-centric pixel coordinates. In trajectory forecasting, the states of the road-agents are in the target-centric coordinates. Technically, the spatial inputs can come from any sensor outputs (e.g., lidar, camera, or map), but we use lidar information only since that is what the PRECOG-Carla dataset provides. We customize hyper-network to consist of three components: (1) a social module, (2) a spatial module, and 3) a time module. The outputs of the three modules h^S, h^Ω, h^T are concatenated and fed into an MLP, which outputs θ, as shown in Figure 3.15.

The social module is designed using Long Short Term Memory (LSTM) [97] or Attention Modules [71]. The module takes $S^{A_k}_{t-\tau:t}$, the historical states of a road-agent in the scene and encodes temporal dependencies and trends among the state parameters. We define $S^{A_k}_t := [x^{A_k}_t, y^{A_k}_t, sin(\theta^{A_k}_t), cos(\theta^{A_k}_t), v^{A_k}]$. We use two LSTM modules; one for encoding the hidden states of N road-agents ($h^{A_{\forall k \neq AV}}$) and the other for encoding the hidden state of AV ($h^{A_{AV}}$). This is because AV is positioned at (0,0) and others somewhere else. Additionally, we use attentions $h^{atn} = Attention(S^{A_{\forall k}})$. The resulting output is the concatenation of the three; $h^S = [h^{A_{AV}} \oplus h^{A_{\forall k \neq AV}} \oplus h^{atn}]$.

The spatial module takes in the processed spatial data, which is denoted as Ω, from external perception modules. Our spatial module is based on convolutional neural networks. We use residual connections to enhance the performance of the convolutional operation of the CNN. Since the backbone network works with Cartesian (x,y) space and pixel (image) space, we use coordinate convolution layers [95] to strengthen the association between the 1D numerical coordinates and 2D image data. Overall, the spatial module consists of 4 CNN-based encoder blocks, and each encoder block consists of 5 coordconv layers with residual connections, max-pooling layers, and batch-normalization layers. The output latent variable is denoted as h^Ω.

Lastly, the time module adds the forecasting time $\Delta t \in \mathbb{R}^1$, i.e. the time difference between the forecasting time of interest and the present time. To increase the contribution of the time condition, we apply an MLP that outputs h^T.

3.10.3 POM Forecasting Results

In this subsection, we present experimental results for POM forecasting. As mentioned earlier, we use HCNAF to produce POM forecasting. CVAE-H is leveraged for trajectory forecasting, which is presented in the next sub-section. We divide our evaluation into qualitative (visualizations) and quantitative evaluations (NLL).

Qualitative results for POM forecasting with HCNAF

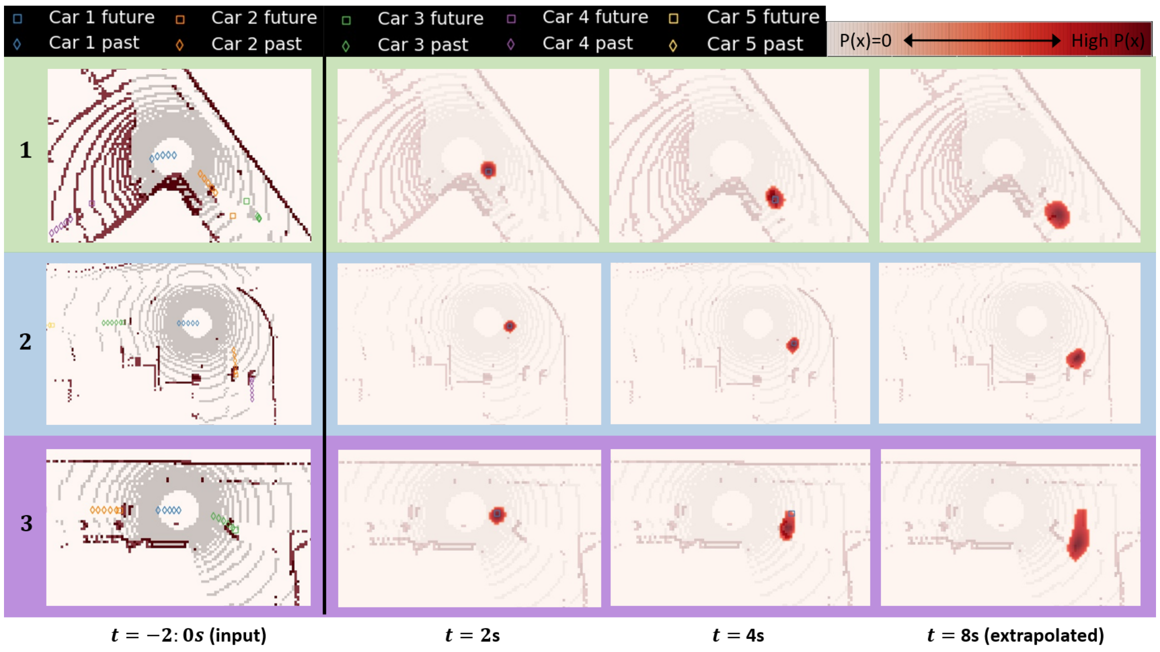


Figure 3.16: The first 3 examples of POM forecasts on the town01 testset via HCNAF. The resulting forecasts indicate that HCNAF is *spatially contextual*.

Qualitative results are presented in Figure 3.16, 3.17, and 3.18. The figures visualize the resulting POM forecasts on nine samples of the Town01-test dataset. Each figure consists of four columns. The first column of the figures represents two seconds history of the road-agent positions ($X^{A_{\forall k}}_{-2:0s}$) that are used as the input to the models along with the lidar data. The second, third, and fourth columns depict the POM outputs of our prediction model for the ego-car depicted as blue. The second and third columns each depict the POM forecast at $t = 2s$ and $t = 4s$ into the future, with overlaying ground-truth positions. Since the dataset only has data from $t = -2s$ to $t = 4s$, HCNAF was only trained on $X^{A_{\forall k}}_{-2:4s}$. Although HCNAF was never trained on $X^{A_{\forall k}}_{t>4s}$, we test HCNAF's extrapolation capability by asking HCNAF to output POM for $t = 8s$, which is a challenging task for any data-driven methods. We report the results in the fourth column of each figure.

Figure 3.16 describes a set of examples that shows HCNAF is *spatially contextual.* In example 1 (test-#116), the blue car (i.e., subject of the prediction) enters a 3-way intersection. HCNAF uses the road topology encoded in the lidar data and correctly forecasts POM at all times along the road boundaries. In example 2 (test-#344), HCNAF is tested if it takes into account the geometry of the curved road. Note that the input trajectory alone does not contain information about road topology. HCNAF successfully makes predictions that are strong evidence that HCNAF is spatially contextual. Again in example 3 (test-#1121), the input trajectory alone does not inform spatial contexts to the model. Nevertheless, HCNAF understands the topology of the 3-way intersection captured in lidar data and produces a multi-modal POM; one for passing through the intersection, and the other for turning right.

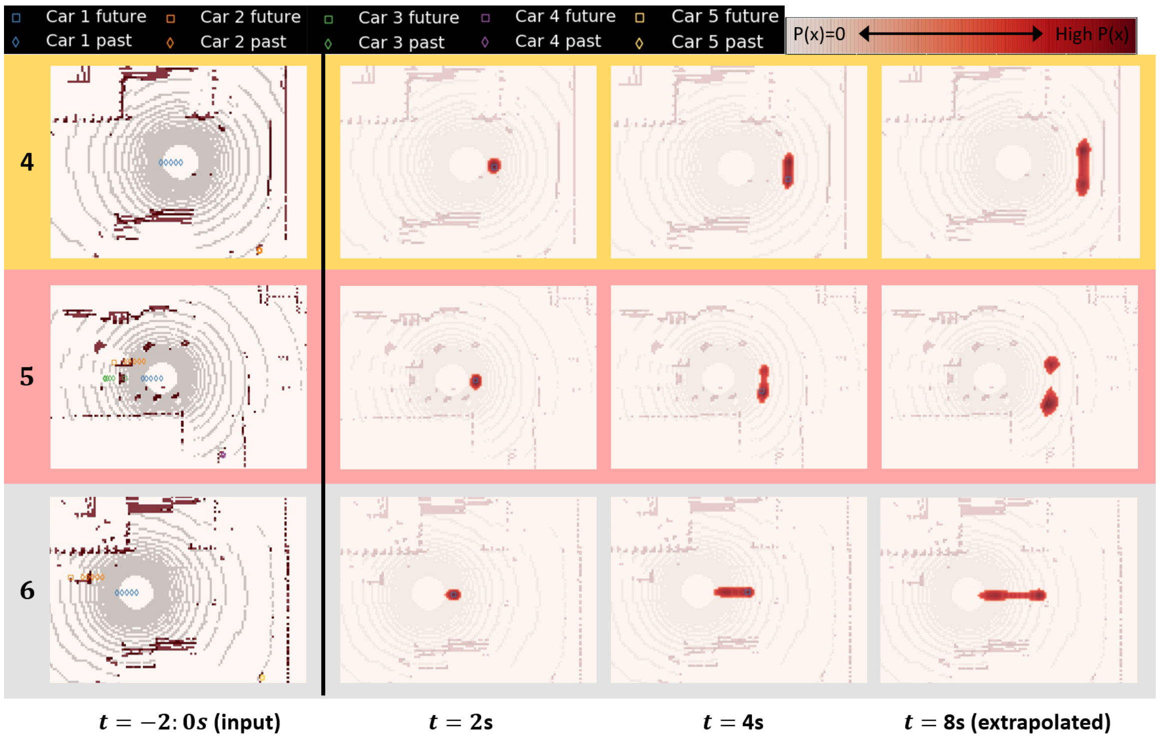


Figure 3.17: The second set of POM forecasts. They suggest that the HCNAF is not only spatially contextual, but also *multi-modal.*

Figure 3.17 describes next three examples from the test set. Example 4 (test-643) depicts a 3-way intersection environment where the blue car is entering the intersection. POM forecasts capture that there exists two natural options (or modes); left-turn & right-turn. Example 5 (test-1252) depicts a 4-way intersection. HCNAF leverages the spatial context and outputs the POM with three modes (left, right, straight), similar to the example 4. Lastly, example 6 (test-640) is another 3-way intersection. The difference from the example 4 is that the location of the blue car. Unlike example 6, the blue car in the example 4 is located at the back of the intersection and there is another car (yellow) waiting for the queue. HCNAF recognizes the subtlety and predicts that there are two possible modes for the blue car. The first is to slow down and yield to the yellow car and the second mode is to continue to move. All three examples in Figure 3.17 empirically show that HCNAF predictions are *multi-modal*.

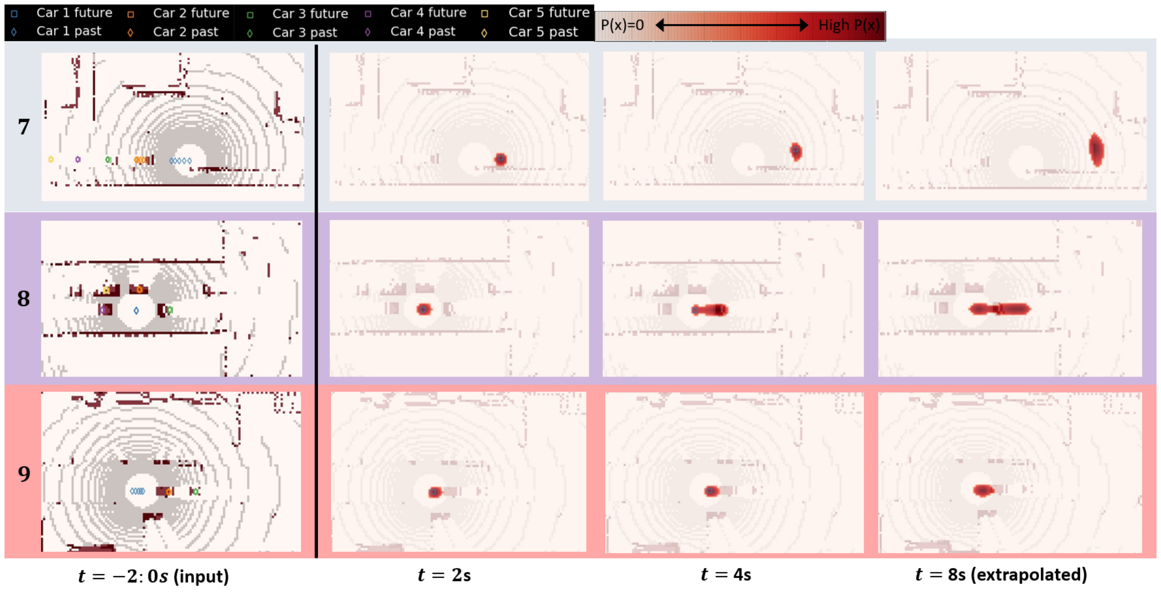


Figure 3.18: Continuing examples (7 through 9) of the POM forecasts. The predictions suggest that HCNAF is *socially contextual*.

Figure 3.18 portrays the last three POM forecasts that are used to check whether HCNAF is *socially contextual*. Let us pay attention to the difference in the blue car's position in the queue. In example 7 (test-253), the blue car is positioned first in the queue and freely enters the 3-way intersection. Reflecting on this context, HCNAF predicts that the blue car will continue to move in relatively fast, either by turning left or going straight. In example 8 (test-519), the blue car is positioned second in the queue and will not be able to move immediately. HCNAF accordingly captures that the blue car may stay idle or move a little. The resulting POM is multi-modal. In example 9 (test-610), the blue car is positioned third in the queue. Since the two vehicles ahead of the blue car are stopped, the blue car is unlikely to move for the next few seconds. HCNAF considers this social situation and successfully predicts that the blue car is most likely to stay where it is.

Quantitative results for POM forecasting with HCNAF

For the quantitative evaluations of the trained models over all data in the test set, we use likelihood statistics. To be precise, we use *extra nats* $\hat{e}$ of the perturbed data distribution, which is lower bounded by zero, instead of NLL, which is unbounded. As introduced in [25], $\hat{e}$ is a normalized likelihood metric defined as $\hat{e} := [H(p', p_{model}) - H(\eta)]/(T \cdot A \cdot D) \geq 0$ (nats/dim), where $H(p', p_{model})$ represents the cross-entropy between p' (data distribution perturbed with an isotropic Gaussian noise) and p_{model}. T, A, D each represents the prediction horizon, number of road-agents, and dimension of the road-agent position. $H(\eta)$ denotes the differential entropy of the perturbed data distribution.

Unlike NLL, $\hat{e}$ provides insights on how close the models are to the theoretical limit that is artificially constructed using the perturbed data distribution. It provides more straightforward understanding of the performance of probabilistic prediction models. Note that there is no upper bound of $\hat{e}$ as $\hat{e} \geq 0$. We used the same $\eta = N(\mathbf{0}, 0.01^2 \cdot \mathbf{I})$ as cited, whose differential entropy is analytically obtained using $H(\eta) = 0.5 \cdot T \cdot A \cdot D \cdot ln(2\pi e|\Sigma|)$. We computed $p(x_t, y_t|C)$ over all time-steps available in the dataset. The results are presented in Table 3.4 and Figure 3.19.

Table 3.4: PRECOG-CARLA Town01 Test, 1 agent, mean $\hat{e}$

Method	**Test ($\hat{e} \geq 0$): Lower is better**
PRECOG-ESP, no lidar	0.699
PRECOG-ESP	0.634
CTFP [98]	0.500
OMEN [99]	0.144
HCNAF (ours)	**0.114**
Ground-truth data distribution	0 (i.e., theoretical limit)

Figure 3.19 provides an in-depth analysis about $\hat{e}$ for the two variants of HCNAF models described in Table 3.4, compared to the published performance of the PRECOG-ESP model. AVG indicates the averaged extra nats of a model over all time-steps $n_t = 1 : 20$ (i.e., $\Sigma_{nt=1}^{20} \hat{e}_{n_t}/20$). Note that the x-axis time steps are 0.2 seconds apart, thus $n_t = 20$ corresponds to 4 seconds into the future. As expected, the POM forecasts $p_{model}(X|C)$ are more accurate (closer to the target distribution $p(X|C)$) at earlier time-steps, as the uncertainties grow over time. For all time-steps, the HCNAF model with lidar approximates the target distribution better than the HCNAF model without lidar. Both with and without lidar, HCNAF outperforms the benchmark model, PRECOG-ESP [25], which is a state-of-the-art prediction model based on Normalizing flow and almost qualifies as an ideal prediction model except for the multi-modality. PRECOG-ESP partially achieves the multi-modality, through recursive autoregressive predictions.

It is worth mentioning that there exists works including [54], [100], which used the PRECOG-Carla dataset. However, most of them only reported trajectory-based predictions metrics (MSD, MinMSD, etc), which will be compared in the following section. To the best of our knowledge, the only available benchmark for NLL on the PRECOG dataset is what is presented in this paper (PRECOG-ESP).

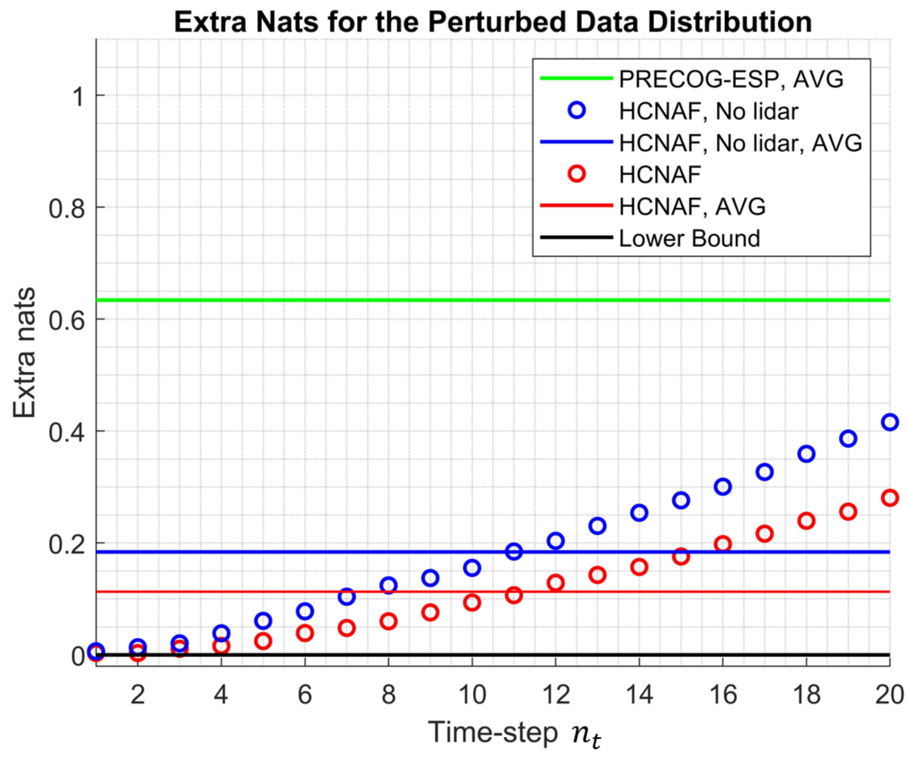


Figure 3.19: Detailed quantitative results for the PRECOG-Carla dataset. Theoretically lower bounded by zero, the metric $\hat{e}$ provides more straightforward understanding of the performance of probabilistic models.

We believe that HCNAF performed better than PRECOG-ESP because HCNAF's expressivity comes from non-linear flow transformations and the hypernetwork framework where the condition terms affect the hidden states of all layers of the hyper-conditioned flow. Note, PRECOG utilizes bijective transformations $f : X \leftrightarrow Z$ that is rooted in affine AF, similar to cMAF (See Equation 3.8). We also believe that the HCNAF's generalization capability is a contributing factor that explains how HCNAF is able to estimate probability densities conditioned on previously unseen contexts.

3.10.4 Trajectory Forecasting Results

This subsection shares the results of the experiments for trajectory forecasting for the PRECOG-Carla dataset. Although HCNAF can generate trajectory samples through the methods explained in Section 3.6.5, it is not as computationally efficient as CVAE-H. For this reason, we use CVAE-H in this experiment. Similar to the previous sub-section, we divide the evaluation into qualitative and quantitative evaluations.

Qualitative results for trajectory forecasting with CVAE-H

Qualitative results from CVAE-H are presented in Figure 3.20 - 3.26. In Figure 3.20 and 3.21, we show how CVAE-H outputs are contextual to spatial features. In Figure 3.22 and 3.23, we show how CVAE-H outputs are contextual to social features. The last set of qualitative examples in Figure 3.24 - 3.26 describe how the number of trajectory samples affect the quality of predictions.

Recall that our prediction outputs are *non-autoregressive*. That is, $S^{A_{\forall K \neq AV}}_{t+\Delta t:T} \sim P_m = \prod_{i=t+\Delta t}^{T} P_m(S_i^{A_{\forall k \neq AV}} | S_{0:t})$ as explained in Section 2.2.2. This explains why we represent the trajectory predictions as independent points, rather than a sequence of points.

In each figure, the left column describes 6 seconds long (2s history + 4s future ground-truth) trajectories of all road-agents and AV. The right column portrays 100 sample predictions per vehicle overlayed with the ground-truth trajectories. As explained before, we perform multi-agent forecasting for all road-agents. The diamond, square, and circle each represents the 2s history, future 4s ground-truth, and prediction samples.

Figure 3.20 and 3.21 visualize four examples of how the resulting predictions are contextual to the spatial features of the environments. The two examples of Figure 3.20, the 329th and 349th example of the town01 test set, portray the 4-second long multi-agent predictions for four road-agents in the scene excluding the AV (blue). The top example is a 4-way intersection environment where the orange vehicle is first in the queue and has three possible modes; left-turn, right-turn, and pass-through. As shown in the right plot of 3.20, CVAE-H captures all three modes in the prediction.

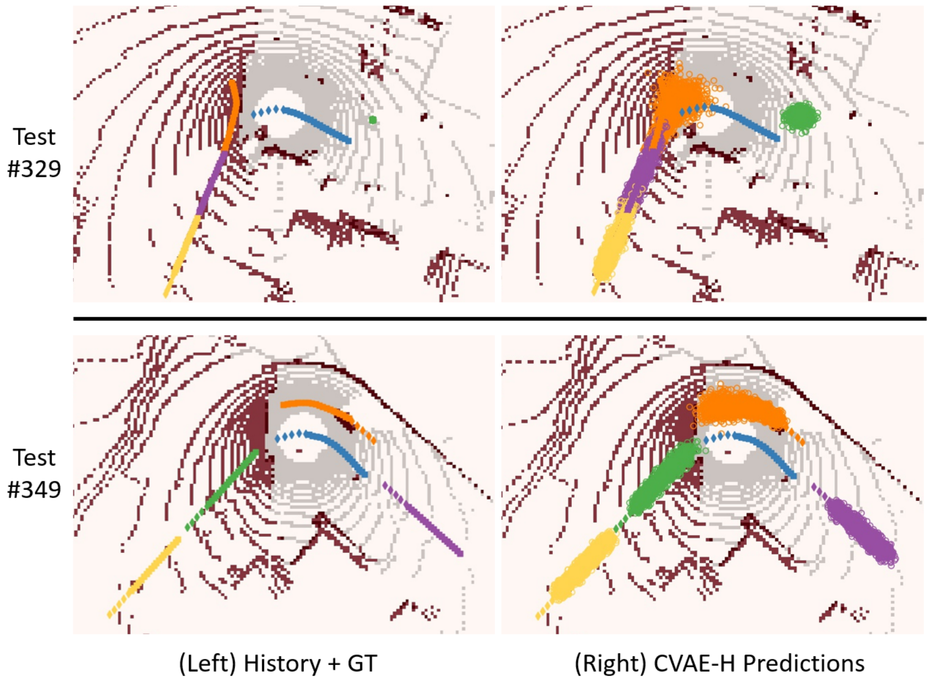


Figure 3.20: The first set of examples qualitatively shows that CVAE-H predictions are contextual to *spatial* features of the environments (i.e., road topologies).

The bottom example is a curved-road environment with five cars driving in the corner of the road. The output prediction for the orange car shows that CVAE-H takes into account the spatial constraint of the curve and makes accurate predictions.

Figure 3.21 presents the second set of examples and supports that CVAE-H is *spatially contextual*. The top example is a 3-way intersection scenario with three exits located at the left, right, and bottom of the plot. CVAE-H not only accurately predicts future positions of all vehicles, but also captures the two distinct modes (left-turn and right-turn) for the purple car. The bottom example is another 4-way intersection environment where the orange car's future motion is quite uncertain given its first 2s history (depicted as diamonds). CVAE-H accounts for this uncertainty and accurately predicts the three possible modes (left-turn, right-turn, and pass-through). The examples in this figure show the *multi-modality* of the output predictions.

CVAE-H considers the social contexts as displayed in Figure 3.22 and 3.23. First,

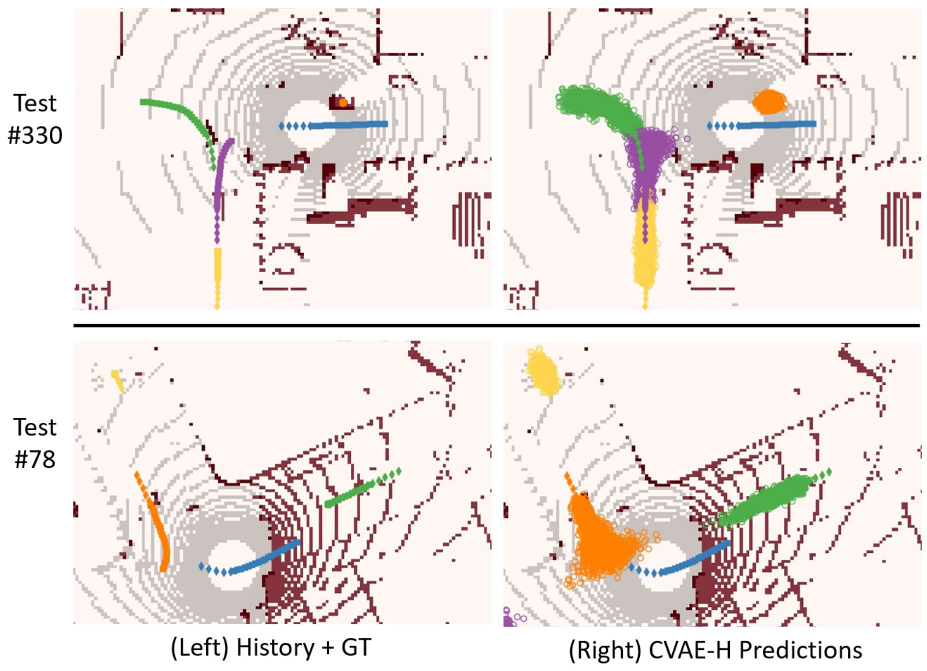


Figure 3.21: Continuing demonstrations of *spatially contextual* outputs from CVAE-H. In addition, they show that the output predictions are *multi-modal*.

let us pay attention to the orange and green cars in Figure 3.22 that describes two 4-way intersection environments. In the top example, the orange car is the first in the queue, whereas the orange car is behind the blue car in the bottom example. CVAE-H understands this social context and accordingly predicts that the orange car in the top example may move in the x-direction, as shown in the elongated uncertainty in the x-direction, whereas the orange car in the bottom is most likely to stay idle. The green cars in both top and bottom examples are last in the queue; CVAE-H understands that there are two other vehicles in front and accordingly predicts that the green car will not move.

Figure 3.23 is the second set of examples, which demonstrates CVAE-H makes *socially contextual* predictions. In the top example, the green vehicle is predicted to stay idle, considering the social situation of the blue car being the first in the queue. Conversely, the yellow car in the bottom example is predicted to move since there is no vehicle in front.

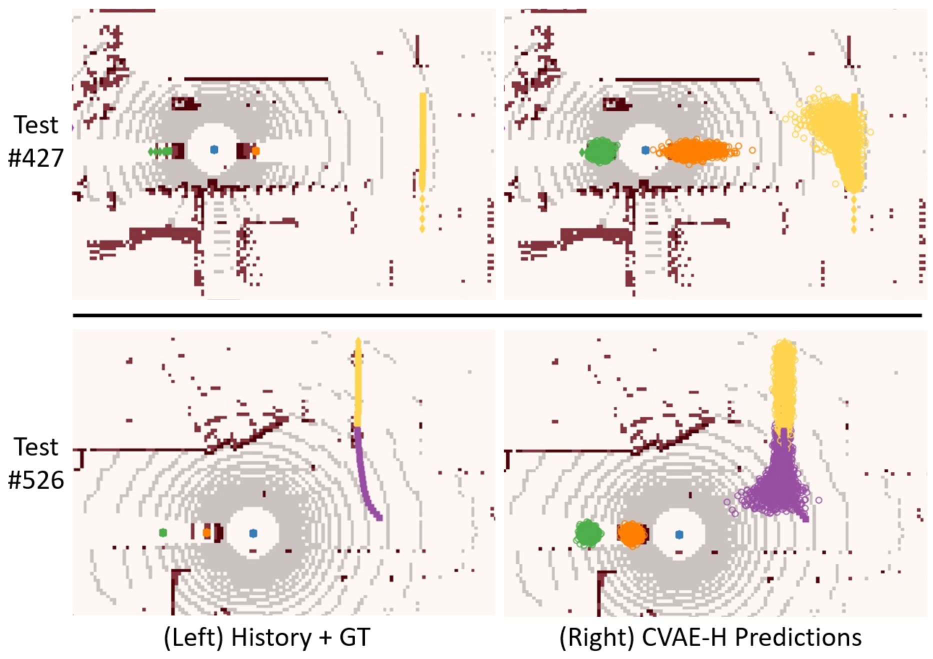


Figure 3.22: Qualitative demonstrations of *socially contextual* CVAE-H outputs.

The fact that CVAE-H not only outputs predictions, but also the probability associated with them makes CVAE-H *probabilitic*. The final output layer of CVAE-H is a Gaussian mixture and the probabilities of the generated predictions are computed analytically. We emphasize again that all qualitative examples presented from Figure 3.20 to 3.26 were produced using a *single* CVAE-H model. This empirically shows that CVAE-H is *general*. In this sense, CVAE-H possesses all four attributes of the ideal prediction model.

Trajectory forecasting v.s. Number of prediction samples

Here, we use three figures to visually show that the quality of the trajectory-based predictions depends on the number of samples. As discussed in Section 2.3, trajectory-based prediction models are used for sampling K possible trajectories, implying that the quality of the prediction largely depends on K. A low K may fail to capture less probable modes. On the flip side, high K means longer inference time and heavier memory requirements.

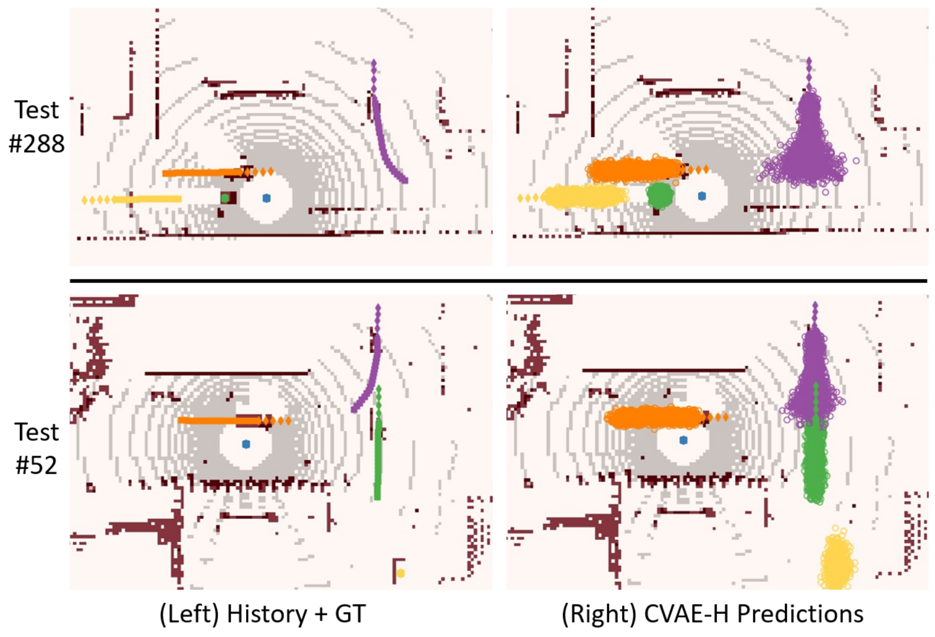


Figure 3.23: Continuing qualitative demonstrations of *socially contextual* prediction outputs.

Figure 3.24 - 3.26 consist of three columns. The left column portrays the 6s long trajectories (2s history + 4s ground-truth future) of the road-agents and AV. The middle and right columns are figures with CVAE-H predictions overlayed. The middle figures only show 3 prediction samples per road-agent, whereas the right figures include 100 prediction samples per road-agent. One could compare how well possible modes are covered with respect to the number of samples.

In Figure 3.24, the green car in the top example has three potential modes; left-turn, right-turn, and pass-through. With 100 prediction samples (i.e., $K = 100$), all three modes are captured. However, with $K = 3$, not all three modes are predicted. Similarly, the bottom example has three modes. While CVAE-H captures all three modes with 100 samples, setting $K = 3$ results in a failure where *left-turn* mode is not included in the sample predictions.

For Figure 3.25, let us pay attention to the green car in the top example and the purple car in the bottom example as their trajectories are uncertain and multi-modal. In the top example, $K = 100$ successfully captures the two modes, however,

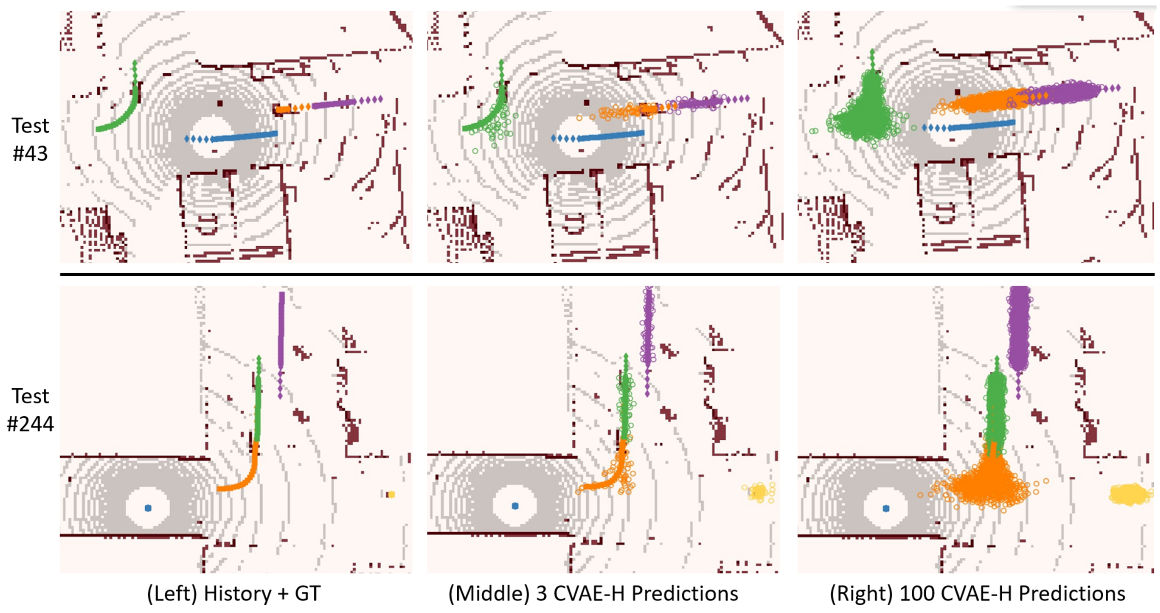


Figure 3.24: The first set of examples showing how the number of prediction samples affects the quality of the predictions.

$k = 3$ fails to capture the left-turn mode. Similarly in the bottom example, $K = 100$ predictions include the multi-modes for the purple vehicle, but $K = 3$ predictions failed to predict the left-turn mode.

Lastly, Figure 3.26 presents the third set of examples that reflect more samples (i.e., higher K) means better chance to capture all modes. We draw the same conclusion as the previous two sets of examples.

Quantitative results for trajectory forecasting with CVAE-H

For quantitative evaluations, we use the minimum mean squared deviation of sampled trajectories from the ground-truth (*minMSD*). MinMSD is a measure of diversity or multi-modality of the model outputs. It is often used for evaluating self-driving prediction models, in particular, generative prediction models that construct probabilistic models with sampling capability [25, 31, 54, 100]. The definition of MinMSD is as follows.

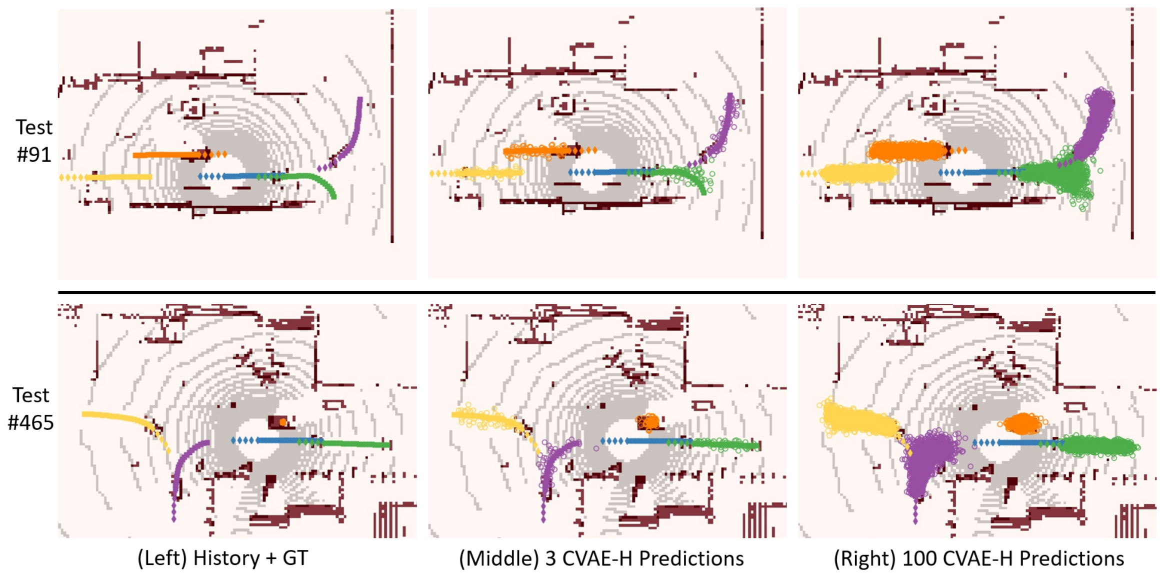


Figure 3.25: Continuing examples show that the sample number affects the prediction quality.

$$\hat{m}_K = min_{k \in 1..K}[||X^k - X^{gt}||^2/T], \tag{3.28}$$

where K, X^k, X^{gt}, and T corresponds to the number of trajectory samples, k-th sample trajectory, ground-truth trajectory, and the prediction horizon, respectively. While there exist similar metrics like minimum average displacement error (minADE) and minimum final displace error (minFDE) [54, 100, 101], we report minMSD to compare our models against the available benchmarks.

Table 3.5 presents the evaluation results against available benchmarks. Our model outperforms most of these prediction models [30, 50, 31, 25, 100], except MFP [54]. While Table 3.5 presented minMSD averaged over all 5 cars, Table 3.6 shares the detailed minMSD numbers for all individual vehicles against available benchmarks.

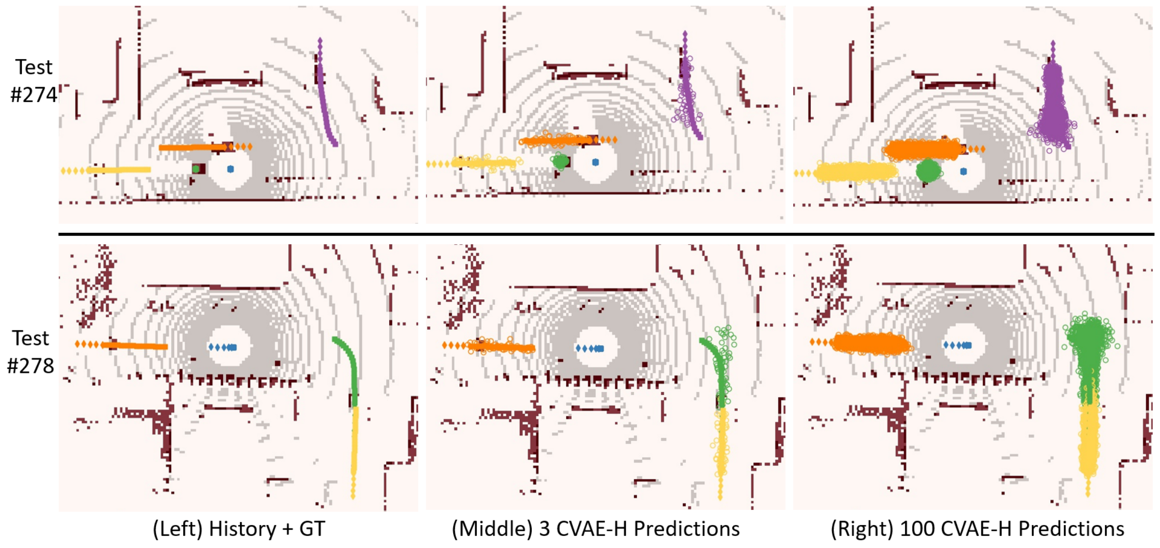


Figure 3.26: Continuing examples reflect more samples corresponds to better chance of capturing all modes.

Table 3.5: Quantitative evaluation of the proposed prediction models using MinMSD with K=12 on PRECOG-Carla town01 testset, averaged over all 5 cars.

Method	**minMSD(m^2) with K=12: Lower is better**
DESIRE [30]	2.599
SocialGAN [50]	1.464
R2P2 [31]	0.843
PRECOG-ESP [25]	0.716
MultiPath [100]	0.680
MFP [54]	**0.279**
CVAE-H (ours)	**0.312**

Table 3.6: The minMSD evaluation results specified for individual agents.

Method	Car 1 (AV)	Car 2	Car 3	Car 4	Car 5
DESIRE [30]	2.621	2.422	2.710	2.969	2.391
PRECOG-ESP [25]	0.340	0.759	0.809	0.851	0.828
CVAE-H (ours)	**0.151**	**0.346**	**0.304**	**0.355**	**0.404**

CHAPTER IV

Planning for Autonomous Driving

Chapter 4 elaborates on the planning part of our solution framework described in Figure 1.3 and 1.4 in Chapter 1 and 2. The planning part corresponds to the second and third steps of the solution implementation detailed in Section 2.2.

The planning task utilizes future states of the environments generated by our prediction models and computes plans optimal and contextual to the environment. We provide an overview of our approach to the planning task in Figure 4.1.

We start this Chapter by discussing important transportation attributes considering the autonomous vehicle is essentially a form of transportation. We incorporate the attributes into designing the objectives of the planner (i.e., reward or cost function) in Section 4.1. Section 4.2 reviews a variety of existing planning algorithms in the field of machine learning, optimal control, and reinforcement learning. In Section 4.3, we make comparisons of the existing planning approaches with particular attention to the *scalability* of the algorithms in the reflection of our research objective (i.e., development of a scalable framework). In Section 4.4, we introduce our learning-based planner and present its mathematical and architectural details. In Section 4.5, we report the evaluation results of the proposed planner against a number of baselines. The evaluation is conducted with respect to the three keywords of the scalability introduced in Section 1.2; *effectiveness*, *generalizability*, and *computational efficiency*

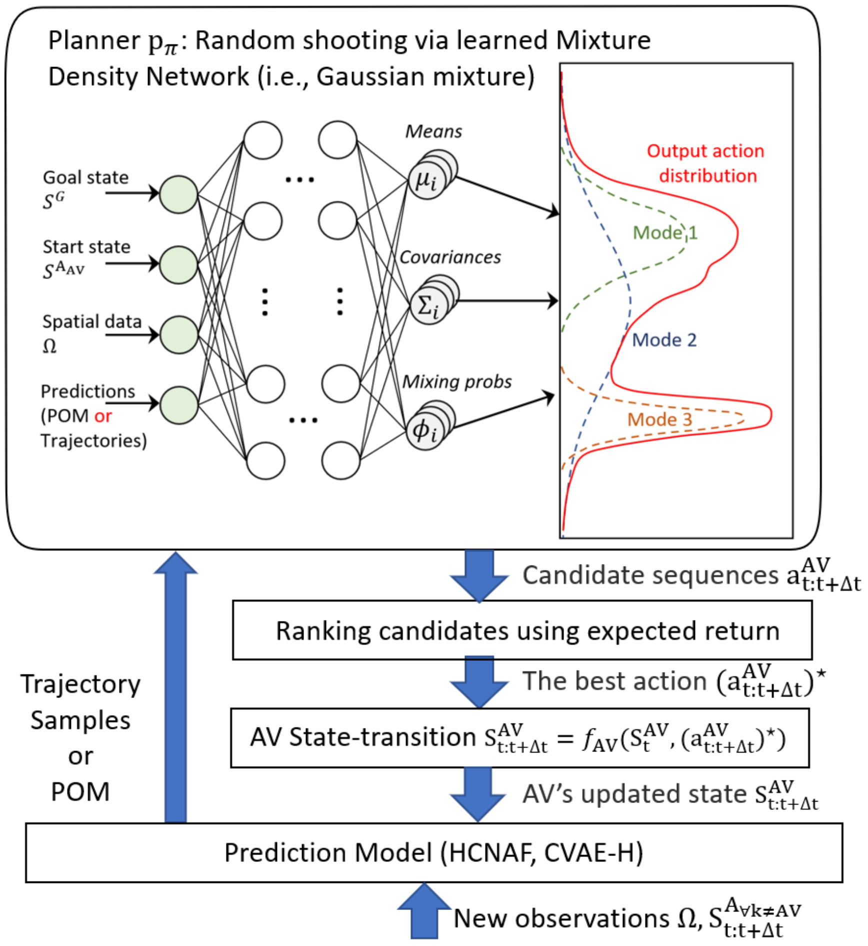

Planner p_π: Random shooting via learned Mixture Density Network (i.e., Gaussian mixture)
Goal state S^G
Start state $S^{A_{AV}}$
Spatial data Ω
Predictions (POM or Trajectories)
Means μ_i
Covariances Σ_i
Mixing probs ϕ_i
Output action distribution
Mode 1
Mode 2
Mode 3
Candidate sequences $a^{AV}_{t:t+\Delta t}$
Ranking candidates using expected return
The best action $(a^{AV}_{t:t+\Delta t})^*$
AV State-transition $S^{AV}_{t:t+\Delta t} = f_{AV}(S^{AV}_t, (a^{AV}_{t:t+\Delta t})^*)$
AV's updated state $S^{AV}_{t:t+\Delta t}$
Trajectory Samples or POM
Prediction Model (HCNAF, CVAE-H)
New observations $\Omega, S^{A_{\forall k \neq AV}}_{t:t+\Delta t}$

Figure 4.1: An overview of the proposed planner.

of the generated trajectory plans. Lastly, in Section 4.6, we perform the planning task with the POM representation and report the evaluation results in comparison to the trajectory representation.

4.1 Planning Objectives

The primary goal of transportation is to move passengers and goods from one location to another. The same applies to autonomous vehicles. An AV should transport passengers and goods from point A (origin) to point B (destination). In this regard, a trajectory planner's primary job is to compute feasible trajectories that connect A and B. We denote the ability to move passengers as *mobility*, which serves as the core objective of our planning problem.

A study investigated a number of transportation systems and commuters' needs [102]. They found that travel time is the most important attribute that commuters care about. Therefore, we can expect that the passengers of AV would prefer shorter travel times. This would be more apparent in urban driving environments where a significant amount of time is wasted in congested traffics [103]. In this regard, *time-efficiency* is an important attribute that AV should possess.

There are several traffic laws all road users should obey. Examples include stopping at red lights, stop signs, and not over-speeding; while a bit of over-speeding is not necessarily harmful, over-speeding by a large gap from the speed limit is harmful. It is natural for human drivers to recognize and follow traffic rules, but it may not be trivial for AV. As training data may contain human driving records that violated a traffic rule, it is necessary for a data-driven planner to be aware of such violations and prevent them from happening. For this reason, *lawfulness* of the autonomous vehicle is an important attribute.

[104] found that ride comfort is also a factor to commuters besides travel time and travel cost. A review paper [105] on passenger comfort discusses factors that create discomforts such as acceleration and deceleration. In this sense, *ride-comfort* is another important attribute for autonomous driving.

The four attributes we introduced here matter only when the safety of the transportation is guaranteed. In other words, safety is a hard constraint of any transportation and it must be guaranteed. The same holds for AV; indeed, researchers have reported that the primary concern of people on autonomous driving is safety [106]. Guaranteeing *safety* is one of the most critical challenges for autonomous driving, especially in urban environments.

The five important attributes, although not exhaustive, of autonomous vehicles are *mobility*, *time-efficiency*, *lawfulness*, *ride-comfort*, and *safety*. In the following, we elaborate on how they are considered in designing the objectives of our planning algorithms.

Often denoted as *reward* or *cost* function, the planning objective provides feedback signals to the AV. As we formulated the research problem using MDP as described in Chapter 2, we incorporate the aforementioned five attributes into the following reward definition.

$$r = r_{goal} + r_{time} + r_{inviolations} + r_{comfort} + r_{safety}.$$

Mobility corresponds to r_{goal} that denotes the goal completion reward and specifies whether the planned trajectories arrive in the vicinity of the destination over the planning horizon. As mobility is the essential attribute of autonomous driving, we prioritize r_{goal} over other attributes in the training of the planner.

Time-efficiency is quantified using r_{time}, which describes how long the AV takes to move from the origin to the destination. The shorter the travel time is, the higher r_{time} gets.

lawfulness is quantified using $r_{inviolations}$, which checks if the traffic rules are obeyed or not. The planner is strongly discouraged from violating the traffic rule since a highly negative reward signal is incurred upon such violation.

ride-comfort corresponds to $r_{comfort}$ that quantifies the degree of ride comfort using the magnitude of acceleration, deceleration, and jerk.

The last attribute, *safety*, is quantified by the term r_{safety} that measures how safe a trajectory plan is. r_{safety} is computed either by checking the probability of AV colliding with an object or evaluating the probabilistic occupancy of the AV's trajectory. The mathematical details of the reward are presented later in this Chapter.

The goal of the planning task is to find the optimal sequence of actions, $(a_{t:T}^{A_{AV}})^{\star}$, which maximizes the expected sum of the discounted rewards as defined in Equation 4.1.

$$
\begin{aligned}
(a_{t:T}^{A_{AV}})^{\star} &= argmax(\mathbb{E}\Big[r_{t:T}(S_{t:T}^{A_{\forall k}}, a_{t:T}^{A_{AV}}, O_{t:T})\Big]). \\
&= argmax(\mathbb{E}\Big[\Sigma_{i=t}^{T}\gamma^{i-t}r_i(S_i^{A_{\forall k}}, a_i^{A_{AV}}, O_i)\Big]),
\end{aligned} \tag{4.1}
$$

where r_t and O_t denote the reward and the POM at time t. γ represents the discount factor. Maximization of the expected return is equivalent to minimizing the cost function J as $(a_{t:T}^{A_{AV}})^{\star} = argmin(\mathbb{E}\Big[J_{t:T}(S_{t:T}^{A_{\forall k}}, a_{t:T}^{A_{AV}}, O_{t:T})\Big])$, however, we adhere to the maximization formulation for the coherence.

Recall that we define a plan as a sequence of actions, i.e., $a_{t:T}^{A_{AV}}$. Given the initial state $S_t^{A_{AV}}$, the trajectory of the AV is obtained simply by iterating the deterministic state-transition function $f_{AV}(S_t^{A_{AV}}, a_t^{A_{AV}}) = S_{t+1}^{A_{AV}}$ from t to T as introduced in Section 2.2.

4.2 Related Works

In this section, we review existing planning algorithms that can be used to solve Equation 4.1. The review is extended to Section 4.3, where we compare these approaches with particular attention to the scalability in the reflection of our research objective. The planning algorithms we discuss here include dynamic programming (Section 4.2.1), optimal control (Section 4.2.2), monte-carlo tree search (Section 4.2.3), value-based RL (Section 4.2.4), policy gradient methods (Section 4.2.5), imitation learning (Section 4.2.6), and random shooting (Section 4.2.7).

4.2.1 Dynamic Programming

First is dynamic programming (DP) which relies on the Bellman's optimality principle [107]. Dynamic programming is the discrete counterpart of Hamilton-Jacobi-Bellman (HJB) equation which gives a necessary and sufficient condition for the optimality of control with respect to a loss function in continuous time [108]. Dynamic programming solves sequential decision-making problems numerically by discretizing state-space of the problem, evaluating the values of the discrete states, and selecting the sequence of the states with the maximum values.

Dynamic programming offers the *global optimal* solution within the discretized state-space. DP also handles various constraints and complex objective functions including non-convex loss functions. Leveraging the advantages, DP has been utilized to tackle different planning problems [109]. Microscopic planning problems such as trajectory planning is the first example. This includes an early work [110] on eco-driving for tracking a predefined drive cycle, a fleet platooning problem [111], and an energy-efficient trajectory planning near signalized intersections [112]. A summary of the applications of DP to eco-driving problems such as eco-cruise control and approaching traffic lights is provided in [113]. The application of dynamic programming for macroscopic planning problems includes vehicle routing problem [114] and vehicle scheduling problem [115].

However, the computation time of DP grows quickly as the state-space gets bigger (e.g., higher discretization resolution and/or longer planning horizon). Known as *Curse of Dimensionality* [116], dynamic programming suffers from scaling the approach to the problems with large state-spaces. For this reason, Dynamic programming is usually used for problems with small state-spaces. However, the planning task in the urban environments involves high planning frequency, long planning horizon, and the presence of multiple static and dynamic objects whose behaviors are highly stochastic. Due to the large state-space of the planning task, the application of dynamic programming is limited.

4.2.2 Optimal Control

Optimal control is a mathematical optimization method that deals with finding a control for a dynamical system over a period of time such that an objective function is optimized [109, 117]. Various applications in science and engineering utilize optimal control to tackle optimization problems. An example is a spacecraft with rocket thrusters as the control and reaching the moon with minimal fuel consumption as the objective (cost) function. Optimal control typically is expressed using a set of equations including differential equations explaining the system dynamics, constraints on the state and control variables, and cost functions.

Optimal control has been widely used across various sequential decision-making problems including vehicle planning problems. Early works solved the adaptive cruise control problem [118], a path-following problem [119] by planning braking and steering, and an obstacle avoidance problem [120]. Applications of optimal control to other vehicle planning problems include [121], which developed an energy-efficient platooning method and [122], which discussed fuel-efficient trajectory planning.

There are various analytical and numerical methods that solve the core optimization problems. Examples of analytical methods include Linear Quadratic Regulator (LQR) for deterministic problems and Linear Quadratic Gaussian (LQG) for stochastic problems. Specifically, LQR and LQG both assume that the state transition, objective function, and constraints to be linear, quadratic, and linear respectively. Additionally, LQG assumes that noise (e.g., prediction uncertainty) is normally distributed. Dynamic programming and Pontryagin Maximum Principle (PMP) can be utilized to solve LQR and LQG problems and provide access to tractable solutions, however, the assumptions are often unrealistic. For the problems that the assumptions do not hold, LQR and LQG can still be deployed, however, they require approximations of the state transition, system dynamics, objective functions, and forms of the uncertainties, which may greatly sacrifice the performance.

Numerical methods such as gradient-based methods, sequential quadratic programming (SQP), and interior point methods are typically used to solve complex optimal control problems. These methods allow optimal control to work with various degrees and forms of objective functions, constraints, and state transition models. While the numerical methods make optimal control more scalable and find decent sub-optimal solutions, they also require a set of assumptions and/or approximations of the state transitions and objective function to functions of lower degrees. These assumptions conservatively estimate uncertainties which as a result make the predictions less accurate and induce the compounding error problem.

4.2.3 Monte-carlo Tree Search

Monte-Carlo Tree Search (MCTS) [123], as the name suggests, is a tree-based search algorithm for discrete decision-making problems. MCTS starts with a top node of a search tree and expands the tree by sampling the search space randomly to select promising actions. At each iteration (i.e., at each node of the tree), an action is selected based on the exploitation and exploration rule of MCTS. Starting from an empty tree, it iteratively selects an action based on an action selection algorithm, performs a state transition with learned state transition models to expand the tree, and evaluates the expected return. Upper Confidence Tree (UCT) [124] is a popular MCTS algorithm that uses a non-uniform action selection rule which encourages searching promising nodes with high expected returns. Like random shooting, MCTS may also involve learning distributions for the action selection process.

MCTS has been notably employed in solving episodic tasks such as Atari games [125] and board games such as GO [126]. Recently, a number of works [127, 128] applied MCTS to decision-making problems for autonomous driving. On the other hand, adoptations of MCTS to continuous problems have been proposed [129, 130]. Extensions to multi-objective MCTS and constrained MCTS also exist [131].

4.2.4 Value-based RL

Value-based RL algorithms refer to the RL algorithms that mainly leverage value $V(s)$ or Q value $Q(s, a)$ to obtain the optimal sequence of actions. As an important concept in MDP, Value estimates how good a state is with respect to the objective of the planning problem. Likewise, Q estimates the value of taking the action a at the state s with respect to the objective. Value-based RL algorithms have shown promising performances across many fields including Atari games [132], robot navigations and manipulations [133], and self-driving [13].

First proposed by [134], Q learning methods are one of the most widely used value-based RL algorithm. Modern Q-learning methods leverage deep neural networks to better approximate Q values. Examples include DQN [132] and double DQN [135]. Value-based methods nowadays leverage ideas from actor-critic methods [136] and further improved the performance [137]. Another popular approach includes a combination of value-based and policy-based RL methods such as DDPG [138] and SAC [139].

4.2.5 Policy-based RL

Policy-based RL is an example of gradient-based planners, which require the rewards and state transition functions to be differentiable in order to obtain the gradients of the objective function. It involves a complex model that evaluates and selects the optimal action sequence using a parameterized model. Once the planner model, or *policy* π, is learned, the remainder of the planning task is to simply run the policy, i.e., $\pi(I) = (a_{t:T}^{A_{AV}})^{\star}$, where I represents the input to the planner. It is often called *policy gradient RL* algorithm and may involve a value function to obtain a better policy (e.g. actor-critic [140]).

Policy gradient methods [141, 142, 139, 143, 144, 9, 10, 145, 146, 12] can be divided into model-free and model-based RL algorithms. As opposed to the model-based RL algorithms that require access to state-transition models or explicitly learn them, the model-free RL methods do not use the models and instead learn policies in trial-and-error fashion. Popular model-free RL algorithms include Natural Policy Gradient (NPG) [142], Soft Actor-Critic (SAC) [139], which is an off-policy model-free RL, Proximal Policy Optimization (PPO) [143], and Distributed Distributional Deep Deterministic Policy Gradient (D4PG) [144].

The aforementioned model-free approaches typically achieve higher rewards than model-based approaches for complex RL problems in tabular and episodic settings. However, model-based RL methods are more sample efficient and can take advantage of physical intuitions about the model (i.e., the dynamical system) by incorporating the prior knowledge about the environments. While there are model-based RL algorithms based on random shooting methods [9, 19, 10], a number of works utilize parameterized networks to explicitly model policies for planning tasks in robotics [145, 146, 12]. For autonomous driving, model-based RL methods are used to tackle decision-making problems by leveraging known system dynamics or learned state-transition models (i.e., the prediction models) [13].

4.2.6 Imitation Learning

Most of the optimal control and reinforcement learning methods including value-based methods, policy gradient methods, actor-critic work under markov decision process (S, a, R, T) formulation. The MDP formulation assumes that agents interact with the environments and receive feedback in the form of rewards. On the other side, there is an approach denoted as *imitation learning* which does not involve rewards (or cost function) and can work independently from the MDP formulation.

Imitation learning (IL) aims to learn a data-driven model by imitating the behavior of an expert such as a human, unlike other sequential decision-making algorithms that learn from reward signals. Imitation learning models learn optimal policies by emulating the demonstrations from the expert. IL is especially useful when it is easier to obtain the expert's demonstrations than to design a reward function that specifies the desired behavior of the agent.

An IL model is typically formulated and trained in the same way that a supervised learning model is formulated and trained. This is why IL is often denoted as *supervised learning for planning*. Given a loss function and a neural network that maps inputs (e.g., states of the environment) to outputs (e.g., actions), the objective of imitation learning is to train the neural network to approximate the policy by minimizing the loss. The simplest form of imitation learning is behavior cloning (BC), which focuses on learning the expert's policy using only supervised learning. An example is a self-driving algorithm that takes sensor information about the environment and learns to produce appropriate accelerations at a signalized intersection scenario [52].

As pointed out by [147], the naive imitation learning (i.e., behavior cloning) often falls into the distribution drift problem. In other words, small errors at each step accumulate over time and the accumulated error may lead the agent to an out-of-distribution state that the agent never encountered during the training. Agents in the out-of-distribution states may not behave optimally and potentially cause safety issues for safety-critical applications like autonomous driving.

There exists a number of resolutions that mitigate the distribution drift problem. One is model predictive control. By re-planning with new observations, MPC can reduce the amount of the distribution drift, however, MPC does not fundamentally solve the problem as the out-of-distribution states remain unchanged. An effective solution is to leverage heuristics in augmenting the data distribution. [14] tackled a self-driving steering control problem using imitation learning and the data distribution was augmented by adding out-of-distribution states and their labels. However, this approach requires a manual design of the data augmentation which is not scalable.

The most popular resolution is to use DAgger [147] which is an algorithm that reduces the discrepancy by continuously expanding the data distribution using the data synthesized by the model policy. However, this also assumes human labeling efforts and thus are not suitable for problems with large-size datasets. Instead of asking humans to label the desirable action, Plato [148] suggests using an optimal control algorithm to automatically produce labels which could potentially solve the problem assuming a fast and accurate annotator is available.

4.2.7 Random Shooting

Random shooting, as the name suggests, is a numerical method where candidate action sequences are randomly sampled from a probability distribution such as a uniform or Gaussian distribution. The sampled action sequences are then evaluated on their expected returns to determine the best action sequence with the highest expected return. As opposed to dynamic programming and optimal control, which obtain the optimal action sequence by solving the core optimization problems, random shooting does not formulate such optimization problems and thus obtains the optimal action sequence in a relatively simple procedure.

Random shooting can be used both in optimal control and reinforcement learning frameworks. It may run either in an open-loop fashion where the entire optimal action sequence is executed at once or in an MPC-based closed-loop fashion where only the first action a_t of the optimal sequence is executed with the repetition of the entire process to find subsequent actions.

Similar to *Deterministic Shooting* that produces deterministic sequences using heuristics or rules, random shooting may be able to generate solutions that are computationally cheaper than the methods that involve optimization problems such as DP and optimal controls. Random shooting may involve learning (i.e., gradient-based training) distributions for the sampling process to increase the performance as well as computation efficiency. This is especially effective for high-dimensional problems with long planning horizons.

Taking advantage of the computational benefit, a number of works utilized random shooting method for designing planners for robotics [9, 19, 10, 11] and for vehicles [8]. Classified as model-based RL algorithms, MBMF [9] used random shooting to solve high-dimensional robotics planning tasks such as Mujoco locomotion planning problems [149], PDDM [19] solved high-dimensional manipulation tasks, PETS [10] used an ensemble of random shooting-based planners to tackle various planning tasks in robotics, and PlaNet [11] showed how a random shooting-based planning method is effective at solving control problems that take high-dimensional images as inputs. On the other hand, [8] utilized random shooting as a vehicle planner in a traffic weaving scenario.

Most of the recent works [9, 19, 10, 11] applied Cross-Entropy Method (CEM) [150] to refine the sampling distributions parameterized via deep neural networks. As opposed to vanilla random shooting, CEM continuously updates the distribution to obtain better action sequences. In this sense, it is closely related to the policy gradient method introduced earlier in this section, however, random shooting methods are usually employed in model-based RL settings.

4.3 Comparisons of Planning Approaches

In this section, we investigate the advantages and disadvantages of the approaches presented in the previous section and select the most suitable approach to our planning problem. Recall our research objective, which was introduced in Section 1.2, is to build a *scalable* prediction and planning framework. In the reflection of the research objective, we pay special attention to the three keywords of the scalability to find the most appropriate planner. The first keyword is *effectiveness* that refers to the quality (i.e., optimality) of the trajectory plans generated by the planner. The second keyword is *computationally efficiency*, which refers to fast inference speed and low memory requirements. The third keyword is *generalizability* that describes an ability to compute near-optimal actions in a variety of different environments.

4.3.1 Traditional Approaches VS Data-driven Approaches

We first categorize the planning approaches we reviewed in Section 4.2 into two groups, depending on whether an approach relies on data-driven techniques or not. The first group is traditional approach which leverage either heuristics or control theories rather than data-driven techniques.

Heuristic approach is the first example of the traditional approach. This includes rule-based models such as constant velocity, constant acceleration, and deceleration model which could be useful for approximating trajectories for vehicles running on highway, departing from an intersection, and approaching an intersection. Physics and heuristic-based models are other examples. For instance, Intelligent Driver Model (IDM) [46], Newell's car-following model [151], and Gipps' model [152] are mathematical models which describe the car-following behavior of human-driven vehicles. While the heuristic approach is computationally very efficient and easily interpretable, it typically works under strong assumptions about the environments and assumes the nominal behavior of human drivers. That is, the approach is not generalizable nor effective for various scenarios of urban driving.

Dynamic programming is another example of the traditional approach. DP finds the global optimal solutions by solving discrete optimization problems. The computation time of DP can be reasonably fast when the problem is deterministic and the state space of the environment is small. However, the computation resource quickly increases as the problem becomes larger (i.e., Curse of dimensionality [109]). Unfortunately, the presence of uncertainties and a high replanning frequency rate enlarge the state space of urban driving environments. In this regard, DP is not best suited for our planning problem.

We categorize optimal control algorithms as the first group. Optimal control provides decent sub-optimal solutions and is typically more computationally efficient than DP. However, it often requires a set of simplifying assumptions to solve the core optimization problems computationally efficiently. Examples of the simplifying assumptions include approximating constraints to convex functions and simplifying uncertainties to simple analytical distributions such as standard normal distribution. The effectiveness and computational efficiency of the solutions come with the expense of the accuracy of the model as the simplifying assumptions may inaccurately approximate the environment and planning objective. Model predictive control could be employed to prevent the approximation to deviate from the actual system and environment. However, the generalizability of the solution is still questionable since optimal control assumes the form of uncertainties and may over-simplify the environments.

On the other hand, the second group of algorithms primarily utilizes data-driven techniques (i.e., reinforcement learning). This includes imitation learning, value-based, policy-based RL, MCTS, and random shooting.

Reinforcement learning algorithms or data-driven methods work with minimal assumptions & approximations about the environment and are thus usually more generalizable than the traditional approach. Specifically, RL algorithms are compatible with diverse representations of the environment. RL can work with diverse forms of uncertainty representations and allows uncertainty-aware decision-making by nature. By leveraging the high representation capability of deep neural networks, RL algorithms are able to approximate complex environments and/or complex decision-making models. In addition, many RL algorithms can perform inference in real-time with the aid of graphic processing units via parallel computing.

The drawbacks of RL algorithms do exist; enforcing hard constraints is achieved indirectly through a reward function. A standalone RL algorithm does not guarantee hard constraint inviolations which could make the system brittle to safety-critical scenarios, requiring an external safety validation [153] or a backup solution that may not be near-optimal, but may guarantee the safety. Some RL algorithms such as policy-based RL and imitation learning may require the differentiability of the model and objective function, however, they are relatively much softer constraints than assuming specific forms of the environment and planning objective. Nevertheless, the advantages of RL are compelling, especially to our planning task where the environment is highly complex and stochastic and the reward function is non-convex. In this regard, reinforcement learning is suitable for diverse urban driving environments.

4.3.2 RL Methods as The Scalable Planner

We discuss the RL algorithms reviewed in the previous section in the perspective of the scalability to select the most suitable planning model for our research objective

First is monte-carlo tree search. While MCTS has solved complex episodic and tabular decision-making problems, it has several drawbacks. The first drawback is that MCTS does not have an explicit evaluation function nor enforce state constraints. Therefore, MCTS needs a modification to accommodate the reward function (e.g., multi-objective MCTS and constrained MCTS [131]). Secondly, MCTS is not as generalizable as other RL approaches. In other words, MCTS is best suited with a specific task [127, 128] and/or episodic problem with well-abstracted environments such as the game of GO [126]. For the non-episodic problems or dynamic problems where the agent often encounters new environments, the MCTS has to perform explorations to estimate the values of the new states and find optimal trajectories. Lastly, MCTS is computationally more expensive than other RL approaches. Unless the states are abstracted well, MCTS quickly becomes computationally intractable for tasks with large state space since it consumes a tremendous amount of search to visit and estimate values of the new states. Parallelization of the search is another challenge that must be addressed to implement MCTS for any planning problem. For these reasons, MCTS is not best suited for our planning problem which requires a scalable planner.

Second is value-based RL which mainly consists of Q-learning and its variants such as DQN [135]. Value-based RL algorithms are uncertainty-aware just like other RL algorithms which are formulated using MDP which captures the probabilistic nature of the environment. However, it may not be the most computationally efficient or generalizable approach. This is because value-based methods assume discrete state-space whereas our planning problem involves continuous state and action space. Discretizing the continuous state-space may cause Curse of Dimensionality. While deep neural networks can serve as functional approximators and mitigate the problem, value-based RL approaches assume strong models that have interpolation and extrapolation capabilities, which may be difficult to obtain. Considering that the state space of our problem changes depending on the environments (or scenarios), value-based methods may have limited generalizability. Another limitation is that value-based RL methods require Markovian assumption to be hold. Since our planning problem takes the high-dimensional input (i.e., the BEV lidar map) and is non-markovian, value-based RL is not the best option.

Third is policy-based (policy gradient) RL methods that learn a policy directly. As policy-based RL methods build on action spaces rather than state spaces, they are more generalizable to new environments with different state spaces but the same action space (e.g., self-driving in urban areas). Since policy gradient methods handle continuous action space by nature, they do not require the discretization of state-space. In this sense, they are computationally efficient. However, model-free policy gradient methods, especially vanilla policy gradient methods [141, 142], are known to have high variance and be more sample inefficient than valued-based methods [154]. There exist different resolutions such as PPO [143] which relies on a clipping in the objective function. Other solutions such as D4PG [144] and SAC [139] leverage parameterized critic networks and thus may have the same limitations as the value-based RL methods. While recent model-free policy gradient methods perform well in complex and stochastic problems [143, 139], they are still less data-efficient and more black-box compared to model-based RL methods.

On the other hand, model-based policy gradient methods can incorporate prior knowledge about the environments into the learning. Given an accurate model of the environment obtained either from mathematical modeling or data-driven techniques, model-based policy gradient methods have shown competitive performances against other RL methods [10, 11, 19]. In this regard, model-based policy gradient RL best qualifies as the scalable planner for autonomous driving in urban areas.

Among model-based RL approaches, random shooting approach [9, 10, 19] is arguably the most interpretable and computationally efficient approach. Random shooting leverages a simple and interpretable distribution such as Gaussian distribution as the policy. Random shooting obtains desired action sequences by sampling the policy and selecting top-k action sequences that correspond to the highest expected returns. The fact that the computation is highly parallelizable and cheap makes random shooting approaches computationally efficient. Besides, it is a gradient-free method, which allows a more flexible design of the reward function compared to gradient-based methods. This means that the reward function can incorporate complex and non-differentiable uncertainties in the environment. A naive random shooting approach may not be scalable as it may take longer to obtain meaningful action sequences for different environments. Deep neural networks can be leveraged to parameterize the random shooting distribution and encode the contextual information about the environment into the distribution. In addition, cross-entropy method (CEM) [150] can be applied to optimize the parameterized policy. Upon successful training, random shooting policy should be uncertainty-aware, effective, and able to generate action sequences contextual to the environment while enjoying the computational efficiency of the vanilla random shooting. For these reasons, we choose the random shooting approach as our planner.

4.4 Planning Algorithm Overview

In the previous sections, we introduced the planning objective, reviewed existing planning approaches, and compared them to find the best-suited approach for our planning problem. In this section, we introduce our learning-based planner, *random shooting via learned Gaussian mixture*, and its mathematical and architectural details. In addition, we describe how the planner fits in the proposed framework described in Figure 1.3 and 1.4.

4.4.1 Prediction Model as The Simulator

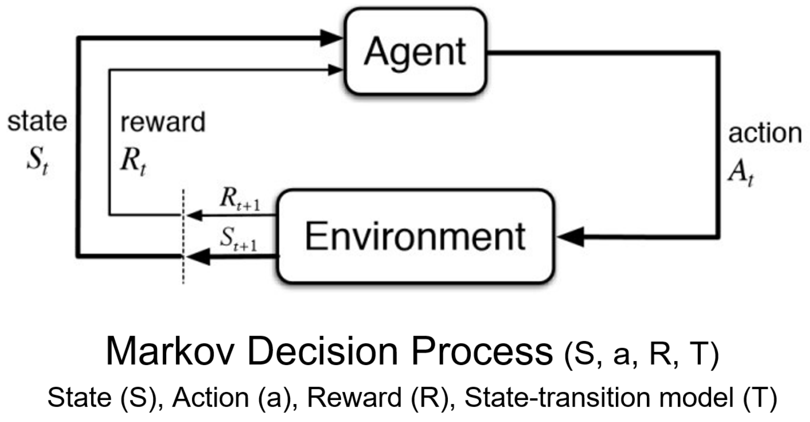

Agent
state S_t
reward R_t
action A_t
R_{t+1}
S_{t+1}
Environment
Markov Decision Process (S, a, R, T)
State (S), Action (a), Reward (R), State-transition model (T)

Figure 4.2: Reinforcement learning is modeled using MDP.

As illustrated in Figure 4.2, RL is concerned with how an intelligent agent takes actions in an environment in order to maximize the feedback (i.e., reward). The same applies to the planning problem for autonomous driving; the planning algorithm of AV, which is an intelligent agent, interacts with the stochastic urban driving environments by taking actions, receiving reward signals, and obtaining new observations of the environments. The agents are trained through iterations of the interactions with the environments.

One could train RL agents in the real world, however, real-world experiments are often very expensive and time-consuming. Moreover, running a planner in real-world environments, especially a planner whose performance was not validated, may invoke safety-critical events. For these reasons, the environment is typically reproduced using a simulator that approximates the real-world environment. In this regard, the performance of the intelligent agent is subject to the fidelity of the simulated environments. The fidelity refers to how close the simulated environments are to the real world. This is true for most RL approaches excluding non-MDP algorithms like imitation learning.

As opposed to planning problems with relatively simple environments (e.g., Atari games), the planning task for autonomous driving involves much more complex environments. This is why it is challenging to create a simulator that approximates behaviors of diverse road-agents under various urban driving scenarios to the high fidelity. There exist open-source urban driving simulators such as Carla [2] and traffic simulators like SUMO [155] which provide a convenient way of simulating the urban driving environments without any safety concerns.

While one can leverage the aforementioned urban driving simulators, they are not light-weighted enough for our planning problem. To be precise, training an RL agent in a simulator assumes that the auxiliary tasks and background jobs to run the simulator are operated in addition to the training. To accelerate the training process, we instead train our model on a collection of MDP tuples (S, A, P, R). In this regard, we utilize the collection from the PRECOG-Carla dataset which consists of urban driving scenarios from Carla simulator [2].

There exists one challenge of this approach for continuous problems. No matter how large the collection of MDP tuples is, a planner will always visit the states that were not part of the collection. This is because the collection is finite as opposed to the infinitely large state and action space of the continuous problems. We address this challenge by building a generative model which can accurately reproduce the state transition and interpolate & extrapolate into the unseen states and utilizing it to simulate the environment. Recall that we empirically demonstrated in Section 3.10 that our prediction models accurately approximate the state transitions of the PRECOG-Carla dataset. In addition, we have shown that our prediction models are highly expressive and able to interpolate & extrapolate into unseen states in Section 3.9. Indeed, the approach of first building a state-transition model and leveraging it in the planning task has been utilized in the RL literature [9, 10, 19, 156]. In this regard, we leverage our prediction models as the simulator to train planning algorithms.

4.4.2 Reward Function

In this subsection, we introduce the mathematical formulation of the reward function.

Recall that the reward, which was first introduced in Equation 2.4, is expressed as a function of goal completion, travel time, traffic law inviolation, ride comfort, and safety (see Section 4.1 for the details). We introduce time t to indicate reward at time t as r_t, which is a function of the environment state and action of AV as follows.

$$r_t(S_t, a_t^{A_{AV}}) = r_{t,goal} + r_{t,time} + r_{t,inviolation} + r_{t,comfort} + r_{t,safety}, \tag{4.2}$$

where each term of r_t corresponds to the followings:

$$\begin{aligned}
r_{t,goal} &= w_{goal}\big[(X_t^{AV} - X^{goal})^2 \leq X_{thres}\big], or \\
&= w_{goal}\big[min(1, X_{thres}/(X_t^{AV} - X^{goal})^2)\big], \\
r_{t,time} &= w_{time}\big[(c_{t0} + c_{t1}t)(t < t_{goal})\big], \\
r_{t,inviolation} &= w_{inviolation}\big[c_{vub}(v_t^{AV} > v_{ub}) + c_{vlb}(v_t^{AV} < v_{lb})\big], \\
r_{t,comfort} &= w_{comfort}\big[c_{rc0}a_t^{AV} + c_{rc1}(a_t^{AV} - a_{t-1}^{A_{AV}})\big],
\end{aligned} \tag{4.3}$$

where $w_{goal}, w_{time}, w_{inviolation}, w_{comfort}$ correspond to the weights of goal completion, travel time, traffic law inviolation, and ride comfort reward terms. All weights excluding w_{goal} are non-positive, suggesting that most reward terms are in fact penalties. $c_t, c_{vub}, c_{vlb}, c_{rc}$ each represents the coefficient for travel time, upper speed limit violation, lower speed limit violation, and ride comfort terms respectively. r_{safety} will be described in the later section as its formulation differs depending on the prediction representation (POM vs trajectory).

In the following paragraphs, we explain the rationals behind the reward formulation. First is $r_{t,goal}$ where we provide a strong positive reward signal to the AV for reaching or getting close to the goal location. We use two different formulations. The first formulation described in the first line of Equation 4.3 is a discrete sparse signal which determines if the AV reached the goal by checking if the squared euclidean distance between the AV and goal is under a threshold X_{thres} at any point (i.e., $r_{goal} = any_t(r_{t,goal})$). The second formulation provides a continuous positive signal which amplifies as the AV gets closer to the goal where the signal is computed using the closest distance between the AV and goal (i.e., $r_{goal} = max_t(r_{t,goal})$). We experiment with both formulations and their performances are reported later in this chapter. On the other hand, $r_{t,goal}$ could be extended to include goal velocity match (e.g., $(V_t^{AV} - V^{goal})^2 < V_{thres}$), however, we exclude the speed match from the reward to allow the planner to be more flexible.

The second reward term, $r_{t,time}$, encourages the AV to arrive at the goal sooner than later. This is encoded in the fact that the penalty only applied for the time $t < t_{goal}$ until the AV reached the goal. Depending on how critical the travel time is to the planner, one could adjust the coefficient of the constant and linear terms. For low travel time penalty, c_{t1} can be set to zero.

The third reward term is $r_{t,inviolation}$, which penalizes traffic law inviolations. Each term of $r_{t,violations}$ respectively penalizes over-speeding and under-speeding. This encourages AV to drive in the range of nominal speeds comparable to other road-agents. If AV exceeds the upper bound of the acceptable speed range or travels slower than the lower bound, the penalty is applied.

The fourth reward term, ride comfort $r_{t,comfort}$, is designed as a function of acceleration and jerk (i.e., derivative of acceleration). As we discussed in the previous section, acceleration and jerk are good metrics of passenger ride comforts. Depending on how acceleration and jerk are important to the planner, one can tune the weights and coefficients of the acceleration and jerk penalty terms.

The safety reward term $r_{t,safety}$ is designed to prevent AV to get too close to any of the road-agents. Recall that there are two different prediction representations: trajectory and probabilistic occupancy map introduced in Section 2.3. We note that each representation corresponds to a different formulation of the safety rewards. We spend two sub-sections for elaborating on the safety rewards.

Planning with Trajectory

The safety reward for the trajectory representation is expressed in the following equation.

$$r_{t,safety} \quad = \quad w_{t,safety} \Sigma_{i \in N_{cand}} \Sigma_{k \in K} \frac{p_m(X_{i,t}^{A_k}|C)}{(X_{i,t}^{A_k} - X_t^{AV})^2 + \epsilon}, \tag{4.4}$$

where $w_{t,safety}, N_{cand}, K, p_m, C$ each respectively denotes the safety reward weight, number of trajectory samples, number of road-agents in the scene, the prediction model (i.e., CVAE-H), and the conditioning information that p_m takes in. As the AV gets close to a sample prediction of a road-agent, the safety penalty increases. $p_m(X_{i,t}^{A_k}|C)$ implies that our planner is uncertainty-aware; a more probable sampled trajectory means that a bigger safety penalty is applied. ϵ is used to prevent the safety penalty from getting infinitely large when the AV gets close to one of the road-agents. $w_{t,safety}$ should be carefully selected as a low $w_{t,safety}$ may result in an unsafe planner and a high $w_{t,safety}$ may result in an overly conservative planner.

Recall that we generate N_{cand} prediction samples per road-agent (see Section 2.3), thus the number of collision checks grows with respect to the number of samples and road-agents (i.e., $N_{cand}K$). A large number of prediction samples make the planner more robust against potential safety-critical events, however, it also increases the computational time and possibly makes the planner more conservative.

Planning with POM

The safety term for a planner that leverages POM $p_m(\cdot|C)$ as the prediction representation is defined as follows.

$$r_{t,safety} = w_{t,safety}\big[c_s p_m(X_t^{AV}|C) + c_{sb}\Sigma_{X_t^{'}\sim n(X_t^{AV})} p_m(X_t^{'}|C)\big], \tag{4.5}$$

where p_m is a prediction model which outputs POM (e.g., HCNAF) and $r_{t,safety}$ consists of two terms. The first term $c_{s0}p_m(X_t^{AV}|C)$ simply computes the occupancy probability of the target coordinate $(X_t^{AV}$. This discourages the AV to move to coordinates that are highly likely occupied by one or more road-agents in the scene.

On the other hand, the second term $\Sigma_{X_t^{'}\sim n(X_t^{AV})} p_m(X_t^{'}|C)$ pays attention to the occupancy probabilities of the neighboring coordinates (or cells) $n(X_t^{AV})$ of the target coordinate X_t^{AV}, where $n(X_t^{AV})$ represents a set of the neighboring coordinates of X_t^{AV}. An example of $n(X_t^{AV})$ is a set of 3x3 adjacent coordinates which altogether form a square whose center is X_t^{AV}. This acts as a safety buffer, hence the name of safety buffer coefficient c_{sb}. The idea is to encourage AV to avoid potentially dangerous regions. This becomes useful in a number of instances including when the target cell is predicted not occupied, however, the neighbors are predicted likely to be occupied. This also mitigates prediction modeling errors and helps AV to prevent from falling into the dangerous regions beyond the planning horizon. It should be noted that c_{sb} needs to be tuned properly, as high c_{sb} can make the decision-making algorithm overly cautious.

We draw an important observation about the computational efficiency of the planning with POM. Note that the number of collision checks with POM representation does not scale with respect to number of prediction samples N_{cand} or number of road-agents in the scene K. This allows the planner to scale better with respect to the number of road-agents and degree of uncertainties in the environments since the computation time remains the same.

4.4.3 Planning with Shooting Planners

In this subsection, we present how a shooting-based planner is leveraged. In our framework, a shooting planner performs the trajectory planning in the following steps.

1. Predict future environment states; $X_{t:T}^{A_{k\neq AV}}$ (trajectory) or $p_m(\cdot|C)$ (POM).
2. Generate N_{plan} candidate action sequences for AV $a_{t:T}^{AV}$ using a shooting planner.
3. Compute AV's state trajectories $S_{t:T}^{AV}$ corresponding to $a_{t:T}^{AV}$.
4. Evaluate $(S_{t:T}^{AV}, a_{t:T}^{AV})$ by computing their expected returns and select the top-k k candidate $(a_{t:T}^{AV})^{\star}$ with the largest return.
5. Execute the first few actions $(a_{t:t_1}^{AV})^{\star}$ (MPC) and obtain new observations $S_{t:t_1}$.
6. Repeat Step 1-5 until the AV arrives at the destination.

Step 1 corresponds to the prediction task presented in Chapter 3. Recall that the planner works in stochastic environments. For the trajectory representation, this means that more than one prediction is necessary to cover diverse future possibilities, as discussed in Section 2.3. Therefore, we predict N_{cand} trajectory samples for each road-agent in the scene. In contrast, for POM representation, we simply load the model $p_m(\cdot|C)$ to the memory without generating any sample.

Step 2 focuses on the generation of action sequences of AV. Recall that the action a_t is a two-dimensional vector of acceleration and steering angle. We consider a discrete sequence of the actions over 4s planning horizon and 0.2s period between each action, i.e., $a_{t:T} \in R^{20\times 2}$. The frequency and planning horizons can be configured, however, we use 4s for the horizon and 0.2s as the period to be comparable with the frequency of the dataset.

In both *deterministic* and *random* shooting approaches, a certain number (N_{plan}) of action sequences are produced $[a^{AV}_{t:T}]_{N_{plan}}$ as candidates. In the deterministic shooting approach, $a^{AV}_{t:T}$ is systematically or manually generated from a finite set of actions. An example is to generate candidate action sequences using equidistant grid points (e.g., $a^{AV}_t \in \{-3, -2, -1, 0, 1, 2, 3\}(m/s^2)$). On the contrary, the random shooting approach obtains candidate action sequences by sampling a continuous distribution such as uniform or Gaussian distributions (e.g., $a^{AV}_t \sim Normal(0, 2m/s^2)$).

In Step 3, we compute the future states, which are future positions and speeds, of AV using the candidate action sequences we generated in Step 2. We use the following vehicle dynamics model (i.e., the deterministic state-transition model f_{AV} first introduced in Section 2.2) to obtain S^{AV}_t, the future states of AV, as follows.

$$S^{AV}_{t+1} = A \cdot S^{AV}_t + B_t, \tag{4.6}$$

$$\text{where } S^{AV} = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}, A = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, B_t = \begin{bmatrix} 0.5a^{AV}_t cos(\alpha^{AV}_t)(\Delta t)^2 \\ 0.5a^{AV}_t sin(\alpha^{AV}_t)(\Delta t)^2 \\ a^{AV}_t cos(\alpha^{AV}_t)\Delta t \\ a^{AV}_t sin(\alpha^{AV}_t)\Delta t \end{bmatrix},$$

where Δt denotes the time period between actions. As described, we use the first-order hold update between the time steps (i.e., $X_{t+1} = X_t + \frac{(V_t+V_{t+1})}{2}\Delta t$, $V_{t+1} = V_t + a_t$). During the experiments, we used $\Delta\alpha$, a steering angle change, instead of α as the second component of the action. We found that this stabilizes the training and results in more optimal trajectories.

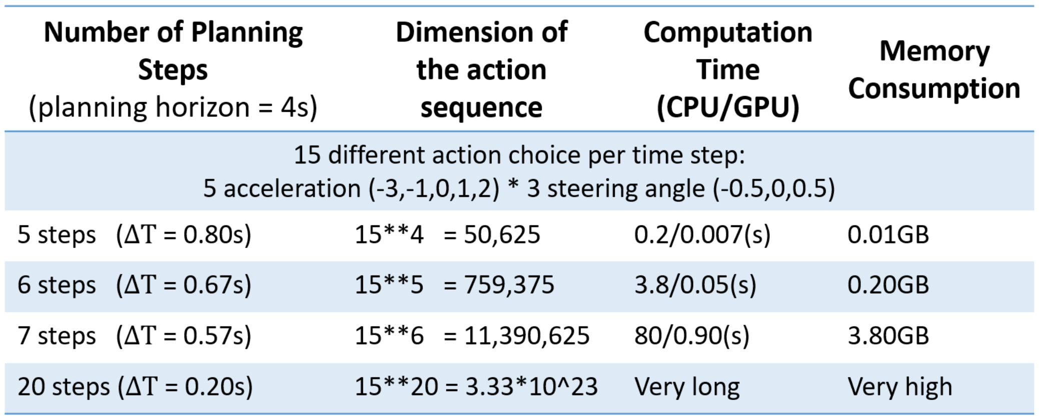
In Step 4, given that the trajectories of AV are obtained, we compute the expected return of all pairs of candidate trajectories & action sequences using Equation 4.2, which is detailed in Equation 4.3, 4.4, and 4.5. For the computation of the safety reward, we use either CVAE-H to output the uncertainties of the generated predictions (for trajectory representation) or HCNAF to infer the occupancy probabilities of candidate trajectories $S_{t:T}^{AV}$ as in $p_m(S_{t:T}^{AV}|C)$ (for POM representation). Once the evaluation is done, we select the top-k action sequences that correspond to the highest top-k expected returns.

Step 5 is the final step of the iteration where we adopt the model predictive control (i.e., receding horizon control). We execute the first m actions $a_{t:t+m\Delta t}^{AV}$ of the best (i.e., top-1) plan $(a_{t:T}^{AV})^\star$ and obtain a new observation of the environment states $S_{t:t+m\Delta t}$. The receding horizon approach, which recomputes the best plans by incorporating new observations, helps minimize the compounding error problem. Once this is completed, it goes back to step 1 again to start a new iteration.

4.4.4 Computational Efficiency of Naive Shooting Planners

The aforementioned procedure works for any deterministic or random shooting planner. However, the computational efficiency and performance of the procedure is separate question. In Section 4.3.2, we briefly mentioned that a vanilla shooting planner can be computationally expensive. In fact, a random shooting planner may require a large number of samples to find meaningful plans and a deterministic shooting planner may incur Curse of Dimensionality. In this subsection, we discuss how critical this problem is and how we resolve this limitation.

Figure 4.3 illustrates how quickly the computational requirements of a naive deterministic shooting algorithm grow. Even a very coarse action grid, which has 15 different action choices per time step, and low planning frequency (e.g., 7 steps, 4s planning horizon, period 0.2s) is unacceptably slow for real-time use.



Number of Planning Steps (planning horizon = 4s)	Dimension of the action sequence	Computation Time (CPU/GPU)	Memory Consumption
15 different action choice per time step: 5 acceleration (-3,-1,0,1,2) * 3 steering angle (-0.5,0,0.5)			
5 steps (ΔT = 0.80s)	15**4 = 50,625	0.2/0.007(s)	0.01GB
6 steps (ΔT = 0.67s)	15**5 = 759,375	3.8/0.05(s)	0.20GB
7 steps (ΔT = 0.57s)	15**6 = 11,390,625	80/0.90(s)	3.80GB
20 steps (ΔT = 0.20s)	15**20 = 3.33*10^23	Very long	Very high

Figure 4.3: The computational efficiency of deterministic shooting quickly decreases as the number of planning steps or resolution increase.

A systematic selection of discrete action and/or state sequences could mitigate the limitation of deterministic shooting. For instance, in a scenario where the AV cruises in the middle of a road with no other road-agent present in the scene, candidate action sequences can simply be limited to $a_t^{AV} \in \{-1, 0, 1\}(m/s^2)$ each corresponds to a moderate break, constant-speed, and moderate acceleration. Another example is a scenario where the AV departs from a stop with no front vehicle present. Reasonable candidate actions would be $a_t^{AV} \in \{0.5, 1, 2, 3\}(m/s^2)$. However, the manual generation of candidate action sequences is not generalizable since there are simply too many variants of environments for urban driving scenarios. Addressing every individual variant of a scenario is not viable. On the other hand, neglecting the subtleties of each variant and solving a few representative scenarios may risk safety and/or may not be the most optimal.

On the other side, random shooting has a few advantages over the deterministic shooting. First, the continuous action samples often find more optimal solutions as they are not subject to the resolution of a grid. Recall that the deterministic shooting typically works with coarse grids to reduce the computation load [8]. Second, random shooting is more comparable with our probabilistic framework. This is because the random shooting works with a probability distribution (e.g., Gaussian), which allows us to leverage machine learning and optimization techniques. Indeed, we leverage the CEM [150] to train our planner which is based on the random shooting method. While the naive random shooting can produce near-optimal trajectory plans for diverse urban driving scenarios provided a large number of sample sequences, it is not the most effective nor computationally efficient approach. This is because it may take infinitely many sample sequences to find a working trajectory plan, given the action space is continuous. We empirically found that naive random shooting typically needs a large number of samples (e.g., hundreds of thousands of samples) to find a trajectory even for relatively simple scenarios like turning in the intersection.

4.4.5 Improving Random Shooting Planner with RL

Let us draw attention to an example of a vehicle moving at constant speed in a stretch of an urban road in free traffic (i.e., no surrounding road-agent is present). In this scenario, it is obvious that the optimal action sequence would not consist of continuous brakings or accelerations. For a vehicle that is approaching an intersection with a stop sign, it is clear that the optimal action sequence will not include excessive acceleration, but rather mild or hard deceleration depending on the speed and location of the vehicle. In this sense, if we know which action sequences are more promising than others based on the contexts of the driving, it will save us the computation resources and help generate a better trajectory plan with higher optimality.

Imagine that there is an oracle distribution that outputs a near-optimal action sequence. Let's assume that the oracle takes the information about the environment. In other words, the oracle is aware of road-agents present in the scene, road topology, the goal position, and the AV's current state. The oracle utilizes these information to produce a set of safe, comfortable, and lawful trajectory plans contextual to the environment state. If we have the oracle, we could simply ask it to output the most promising trajectories, instead of generating and evaluating a very large amount of candidate trajectories. The oracle is essentially the planner we aim to build.

To build such a planner, our idea is to leverage deep learning and introduce model-based RL techniques to the random shooting method. Instead of using a pre-defined Gaussian or uniform distribution, we build a data-driven conditional distribution (i.e., policy) $p_\pi(a^{AV}|S)$ that takes the environment state as the input and outputs contextual trajectories. We model the policy using a deep neural network and train the policy using a loss function that consists of expected return and cross-entropy.

4.4.6 The Proposed Planner: Random Shooting with via Learned Gaussian Mixture

The policy of the proposed planner, $p_\pi(a^{AV}|S)$, is modeled using a mixture density network (MDN) [37]. We choose the core distribution of MDN to be a mixture of multi-variate Gaussian distributions whose parameters include means μ, co-variances Σ, and mixing coefficients ϕ ($\sum \phi_i = 1$). The parameters are modeled using a deep neural network that consists of MLP, RNN/Attention, and CNN to encode spatial and social contexts of the environment.

Figure 4.4 illustrates the network design of the backbone deep neural network of the planner. The input to the planner is a tuple of four tensors S^G, S^{AV}, P, Ω. First, the planner encodes the spatial information about the environment either by running a CNN to obtain the encoding from Ω or by re-using the encoding tensor Ω_{enc} provided by the prediction model (i.e., bypassing the CNN). The CNN is designed using residual connections [70] and coordinate convolution layers [95]. While the former makes the planner heavier and slow, it may be able to encode information useful to the planner. The other three components of the input S^G, S^{AV}, P are each passed to different MLP layers to obtain encoding and passed to a self-attention layer or RNN. Finally, the two hidden tensors are concatenated and go through another

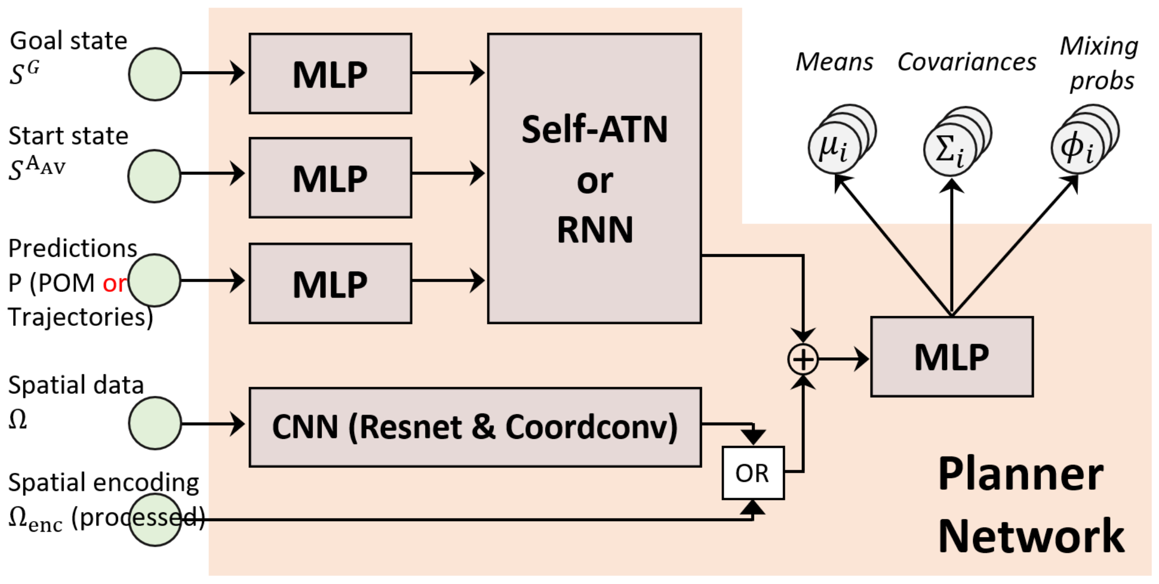


Figure 4.4: The planner network design.

MLP layer which outputs the parameters of the Gaussian mixture.

The training objective is a weighted combination of the expected returns of top-k state-action sequences and cross-entropy between the top-k sequences and the model Gaussian mixture distribution. The cross-entropy maximization between the policy and top-k candidates is denoted as Cross-Entropy Method (CEM) [150] which has been used in a number of robotics planning literature [9, 10, 11, 19].

The training procedure is divided into a couple of steps as follows.

1. Initialize the parameters of the policy p_π.

2. Retrieve a training example from the dataset. Generate a prediction of the environment P (trajectory samples or POM) using the prediction model p_m.

3. Given inputs $S = (S^G, S^{A_{AV}}, \Omega, P)$, sample $p_\pi(a^{AV}|S)$ to generate N_{plan} candidate action sequences $\left[a^{AV}_{t:T}\right]_{N_{plan}}$.

4. Compute AV's state trajectories $\left[S^{AV}_{t:T}\right]_{N_{plan}}$ corresponding to $\left[a^{AV}_{t:T}\right]_{N_{plan}}$.

5. Evaluate the expected returns for the candidate sequences $\mathbb{E}\Big[r_{t:T}(S_{t:T}^{AV}, a_{t:T}^{A_{AV}}, S)\Big]$. Select the top-k k candidate $\big[a_{t:T}^{AV}\big]_k$ with the highest returns.

6. Compute the negative log-likelihood of the top-k candidates -log$p_\pi\big(\big[a_{t:T}^{AV}\big]_k|S\big)$.

7. Back-propagate the loss = NLL (top-k) + $r_{t:T}$ (top-k). Repeat Step 2-7.

We start the training procedure by defining the policy network and initializing the parameters using Xavier initialization [157]. As we discussed in Section 4.4.1, we use our prediction model as the simulator to generate the predictions P. In step 3, the planner p_π takes the current environment state S and generates N_{plan} candidate action sequences. Here, S is a tuple of goal state, start state, spatial data, and the predictions. In step 4, the AV's state trajectories $\big[S_{t:T}^{AV}\big]_{N_{plan}}$ are simply computed using f_{AV} defined in Equation 2.2. Step 5-7 describe how the loss function is computed and back-propagated. The top-k number k and sample plan numbers N_{plan} are important parameters that determines the success of the training. We report how different k and N_{plan} affect the performance of the planner in the next section. It is worth mentioning that the MPC-like approach could be applied to the training on top of the inference. In this case, there would be an additional step where we execute the first few actions $(a_{t:t+\Delta t})^\star$ of the best action sequence, discard rest of the actions $(a_{>t+\Delta t})^\star$, and obtain new observations at $t = t + \Delta t$. This would create an outer loop where Step 2-8 are performed multiple times until the MPC horizon ends.

Let us denote $p_\pi^\star = p_{\pi_{\theta^\star}}$ as the optimal planner that we seek for. For the training set of size N, the optimal policy $p_\pi^\star$, or the optimal parameters of the policy network $\theta^\star$ is obtained as follows.

$$\theta^\star = argmax_\theta \mathbb{E}_{i\in N}\mathbb{E}_{j\in k}\Big[r_{t:T}\big[a_{t:T}^{AV}\big]_j + logp_{\pi_\theta}(\big[a_{t:T}^{AV}\big]_j|S_i)\Big], \tag{4.7}$$

where the reward computation $r_{t:T}$ is detailed in Section 4.4.2. After the training, we perform the inference (i.e., generation of the best trajectory) according to the following procedure.

1. Retrieve a training example from the dataset. Generate a prediction of the environment P (trajectory samples or POM) using the prediction model p_m.

2. Given inputs $S = (S^G, S^{A_{AV}}, \Omega, P)$, sample $p^\star_\pi(a^{AV}|S)$ to generate N_{plan} candidate action sequences $\left[a^{AV}_{t:T}\right]_{N_{plan}}$.

3. Compute AV's state trajectories $\left[S^{AV}_{t:T}\right]_{N_{plan}}$ corresponding to $\left[a^{AV}_{t:T}\right]_{N_{plan}}$.

4. Evaluate the expected returns for the candidate sequences $\mathbb{E}\Big[r_{t:T}(S^{AV}_{t:T}, a^{A_{AV}}_{t:T}, S)\Big]$. Select the top-1 candidate $(a^{AV}_{t:T})^\star$ with the highest expected return.

5. (MPC) Execute the first few steps of the top-1 action sequence, i.e., $(a_{t:t+\Delta t})^\star$. Repeat Step 2-5 until the AV reaches the goal.

We would like to note that the step 1-3 of the inference procedure are identical to the step 2-4 of the training procedure. The difference is that we use the top-1 sample in the inference and do not compute the loss or cross-entropy.

4.5 Evaluation: Generic Urban Driving

In this section, we demonstrate the performance of the proposed planner in the generic urban driving scenarios of the PRECOG-Carla dataset. We train the proposed planner on the town01 training set, validate the planner using the validation set, and evaluate the performance on the test set. We refer readers to Section 3.10.1 for the details of the dataset.

We start this section by first discussing the evaluation metrics in Section 4.5.1. Similar to the evaluation of the prediction models, we evaluate the planner qualitatively and quantitatively. In Section 4.5.2, we go over the baseline planners that we use to compare the performance of our planner. Section 4.5.3 reports the configuration we use to set up the baselines and the proposed planner. Finally, Section 4.5.4 and 4.5.5 present the evaluation results.

4.5.1 Evaluation Metrics

The evaluation of the planner is divided into (1) qualitative evaluation and (2) quantitative evaluation. For the qualitative evaluation, we share a number of figures which visually describe the environments and depict the trajectory plans produced by the proposed planner. An example visualization is illustrated in Figure 4.5.

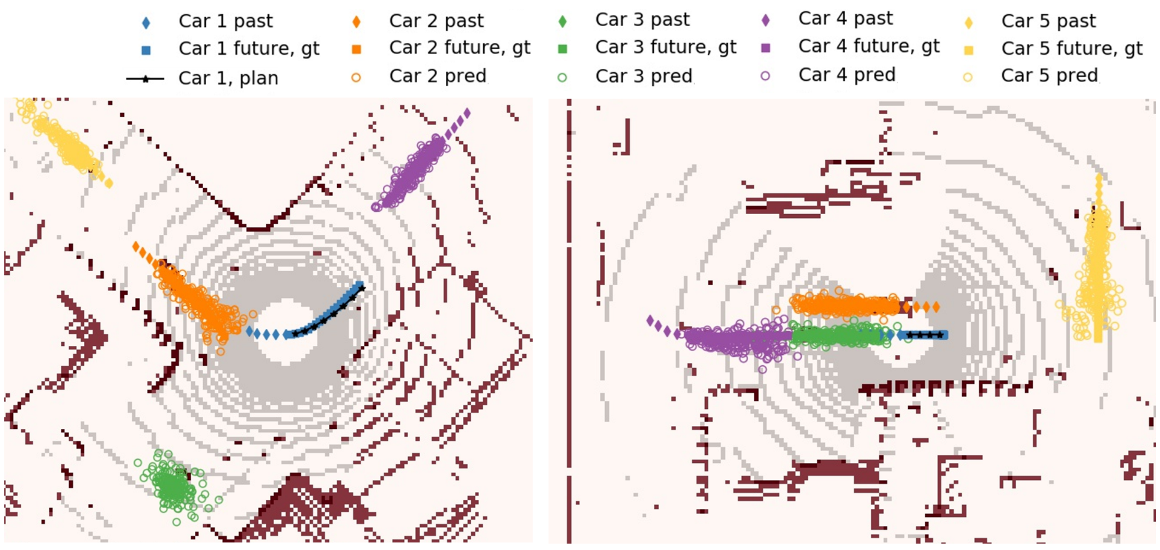


Figure 4.5: Example visualizations of the planning results. A visualization depicts spatial information (lidar), road-agents' historical $X_{t-2:t}^{A_{\forall k}}$ and future (ground-truth) $(X_{t:t+4}^{A_{\forall k}})^{gt}$ states, and predicted road-agent states. The trajectory plan $(X_{t:t+4}^{AV})^{\star}$ from a planner is overlaid on the top of the plot.

As described in the labels of Figure 4.5, the best trajectory plan $(X_{t:t+4}^{AV})^{\star}$ is depicted as a line of black star markers. The predicted road-agents' future states are described using circle markers. Using the visualization, we qualitatively evaluate how well the planner performs in different scenarios including turning, passing through an intersection, waiting in a queue of the intersection, and cruising in the middle of a road.

On the other side, we use the following three metrics for the quantitative evaluation of the planner: (1) optimality, (2) generality, and (3) computational efficiency of the planner. The optimality of the planner is computed using the average of the expected returns over all test set examples. The generality is quantified separately from the optimality. Specifically, we make a box plot of the expected returns and compute the box-plot metrics including the minimum, 25% quartile, and median. Essentially, a generalizable planner that works across diverse environments should have high minimum, 25% quartile, and median values. Lastly, the computational efficiency is measured using the memory requirement and computation time of the planner.

Recall that our research objective is to build a *scalable* framework. As we mentioned a few times, the three keywords of the scalability are effectiveness, generalizability, and computational efficiency. We emphasize that the three quantitative metrics are closely related to the scalability. Precisely, the optimality, generality, and computational efficiency of the planner each corresponds to the effectiveness, generalizability, and computation efficiency for the scalable prediction and planning framework for autonomous driving in urban areas.

4.5.2 Baseline Models

To better evaluate the proposed planner, we test a number of baseline models and compare their performances against the proposed planner. The baseline models include dynamic programming, rule-based models, heuristics, naive deterministic shooting, naive random shooting, and behavior cloning. All baselines and our proposed planner work under the same framework. That is to say, we use the same prediction models to obtain predictions of the road-agents, same system dynamics for AV (i.e., identical state-transition model), and the identical reward structures and weights (i.e., identical feedback function).

Dynamic programming is arguably the most important baseline as it produces the global optimal solution for any environment. The optimality of the DP solution serves as the upper bound of the optimality for any planner. In this sense, we define the *near-optimal* planner as a planner whose expected returns of output trajectory plans are almost as high as the expected returns of the DP solutions. We note that we were only able to discretize the DP's state and action space to a certain degree due to the computation resource limit. To this end, the optimality of the DP solution is only optimal with respect to the resolution of the state and action space, but not the true optimum, which is obtained in the continuous space. The resolution of the grids used to produce DP solution is described in Table 4.1.

Rule-based models are simple baselines that are typically the fastest and most interpretable. The computation requirements are close to none as they do not involve any complex logic to compute an action sequence. The rule-based policies we implemented include constant-speed, constant-acceleration, and constant-braking. The constant-speed refers to a policy $p_\pi(a^{AV}_{>t}|S) = [0(m/s^2), 0(rad)]$ that maintains the initial speed and heading angle until the end of the planning horizon. The constant-acceleration and constant-braking each refers to $p_\pi(a^{AV}_{>t}|S) = [0.5, 0]$ and $p_\pi(a^{AV}_{>t}|S) = [-0.5, 0]$. The rule-based models apply the same actions regardless of the context of the environment.

Naive shooting methods are another baselines we tested. Due to the deterministic shooting's curse of dimensionality (see Section 4.4.4), we only experimented with random shooting. In the naive random shooting, candidate action sequences are sampled using a bivariate Gaussian as follows.

$$a^{AV}_{t:T} \sim N(\mu_{naive\_rs}, \Sigma_{naive\_rs}), \tag{4.8}$$

where $\mu_{naive\_rs} = [2(m/s^2), 0(rad)]$ and $\Sigma_{naive\_rs} = [[2/3(m/s^2), 0], [0, 0.25(rad)]]$.

We experimented with different combinations of the mean and covariance of the Gaussian and we found that the above combination was the optimal combination for the PRECOG-Carla Town01 test set. Since N_{plan}, which refers to the number of candidate trajectory plans, greatly affects the performance of the naive random shooting, we report the results with different N_{plan} values.

There are two heuristic-based models we tested. The first is a goal-conditioned constant acceleration model that we designed to guarantee the goal completion and minimize the ride-comfort penalty. We design the model to automatically compute an output trajectory that starts with the initial state, moves towards the goal position with a constant acceleration, and ends at the goal position precisely at the end of the planning horizon (i.e., $t = 4s$). The trajectory and action are computed using the logic described in Equation 4.9. Note that this model does not guarantee safety as it does not take the positions of other road-agents into account. Nevertheless, the AV is guaranteed to reach the goal without creating any jerk and thus, this serves as a powerful baseline.

$$\begin{aligned} \Delta X^{AV} &= V^{AV}_{t=0}\Delta t + 0.5a^{AV}_t(\Delta t)^2, \\ a^{AV}_t &= -2V^{AV}_{t=0}/\Delta t + 2(X^{goal} - X^{AV}_{t=0})/(\Delta t)^2, \\ a^{AV}_t &= -0.5V^{AV}_{t=0} + 0.125(X^{AV}_{t=4}(= X^{goal}) - X^{AV}_{t=0}). \end{aligned} \tag{4.9}$$

Intelligent Driver Model (IDM) [46] is another heuristic-based model. IDM is a mathematical model which describes a nominal car-following behavior. While IDM also works for free-flow traffic, we found that vanilla IDM does not perform well under free-flow traffic for our dataset. To improve IDM's performance, we use the constant-speed policy when there is no vehicle in front of the AV. In addition, IDM requires labels of the lead vehicle that the AV is following. Since no such label is available in our dataset, we manually check all states of road-agents in the scene if any qualifies as the lead vehicle. The logic which consists of three conditions is described as follows.

$$\begin{aligned}
&c_1 = \dot{X}_t^{A_k} \bullet \dot{X}_t^{AV} > 0, \\
&c_2 = (X_t^{A_k} - X_t^{AV}) \bullet \dot{X}_t^{AV} > 0, \\
&c_3 = X_{min} < X_t^{A_k} - X_t^{AV} < X_{max}, \\
&flag = c_1 \wedge c_2 \wedge c_3.
\end{aligned} \tag{4.10}$$

Behavior cloning (BC) is the last baseline we tested. We organize the planning problem as a behavior cloning problem (i.e., supervised learning with a distance metric) and design the backbone network of the BC planner using the CVAE-H network design. Essentially, the model architecture of the behavior cloning planner is almost identical to the CVAE-H prediction model, except that we use an autoregressive version of the CVAE-H architecture, as opposed to the non-autoregressive CVAE-H prediction model. This is because the planning task requires outputs to be as dynamically feasible as possible, as opposed to the prediction task, which can work with conditionally *independent* trajectories $p(X_{t=0:T}|C) = \prod_{t=0}^{T} p(X_t|C)$ states rather than conditionally *dependent* trajectories $p(X_{t=0:T}|C) = \prod_{t=0}^{T} p(X_t|C, X_{0:t})$.

Another difference is that its learning objective is different from the prediction model. Since our dataset is inherently multi-modal, learning the AV's policy makes the BC policy to output multi-modal trajectories. In this sense, we generate a large number of behavior cloning trajectories, find the trajectories with the mode we are interested in, and select the one with the highest expected return to compute the optimality. We note that our behavior cloning baseline does not involve any additional training techniques such as DAgger [147] or Plato [148] used in imitation learning as most require extensive human labeling efforts or expensive optimal control as a labeler.

Table 4.1: Resolution of the grids used to run the baselines or proposed planners.

Method	Action	Speed	Position	Time
Dynamic Programming	$0.5(m/s^2)$	$2.0(m/s)$	$2.0(m)$	0.4s
Heuristic-based Models	N/A - continuous			0.2s
Rule-based Models	N/A - continuous			0.2s
Naive Random Shooting	N/A - continuous			0.2s
Behavior Cloning	N/A - continuous			0.2s
Proposed Planner	N/A - continuous			0.2s

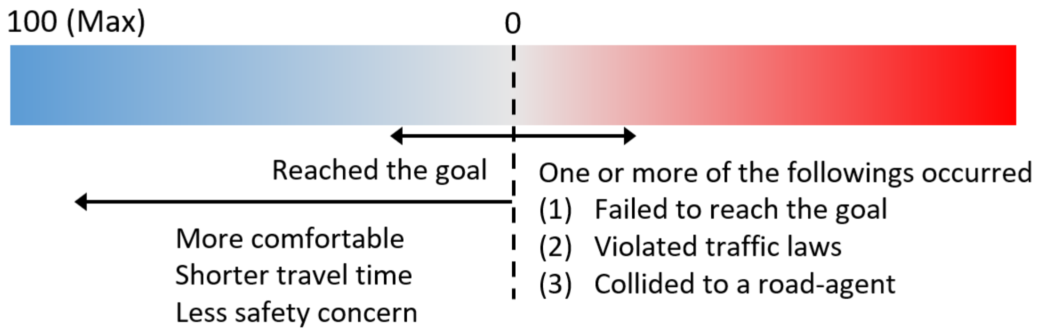

100 (Max)
0
Reached the goal
More comfortable
Shorter travel time
Less safety concern
One or more of the followings occurred
(1) Failed to reach the goal
(2) Violated traffic laws
(3) Collided to a road-agent

Figure 4.6: The optimality of a trajectory plan is quantified using the expected return of the illustrated scale.

4.5.3 Configuration of The Proposed Planner

Recall that the expected return is a sum of discounted rewards at each time step; $r_{t:T} = \sum_{i=t:T} \gamma_i r_i = \sum_{i=t:T} \gamma_i [r_{t,goal} + r_{t,time} + r_{t,inviolation} + r_{t,comfort} + r_{t,safety}]$. In our experiments, we set $\gamma_i = 1$ to put the equal importance for rewards at all time steps.

Figure 4.6 presents the visual description of the scale of the optimality. We set 100 as the upper bound of the expected return $r_{t:T}$. The upper bound is achieved when the AV reaches the goal instantly at $t = 0$ without violating any traffic law or jeopardize safety. Thus the upper bound cannot be reached unless the goal state is exactly the same as the initial state of the AV. Each action that the AV takes, longer travel time, traffic violations, and collision all reduce the expected return by a certain amount. Among the four negative terms (travel time, traffic inviolations, ride comfort, and safety) of r_t, the safety and traffic violations penalized the most

Table 4.2: The weights for the reward terms used in the experiment.

Weight	Value
w_{goal}	+100
w_{time}	-5
$w_{comfort}$	-9 (acceleration), -36 (jerk)
$w_{inviolation}$	-100
w_{safety}	-100 (collision or the distance $\leq$ 1m), or -0 $\sim$ -100 (mild safety concern is pro-rated w.r.t. distance).

strictly. Upon such violation, the reward diminishes by 100, making the expected return always negative regardless of the goal completion. The ride comfort, travel time, and mild safety concerns (i.e., the AV is close to a road-agent, but not close enough to cause a collision) also reduce the expected return, however, the amount of the reduction is relatively small. For example, if the AV successfully reached the goal at the end of the planning horizon with frequent brakings and accelerations, the expected return will still be over zero. Of course, if a trajectory plan can get AV to the goal with minimal use of acceleration & steering, a short travel time, no safety concerns, or traffic violation, the expected return would be close to 100. Table 4.2 presents the weights of the reward terms.

4.5.4 Qualitative Evaluation

The qualitative results of our planner are presented in Figure 4.7 - 4.10. Each figure describes different environments with different goal positions. In each figure, two sample scenarios are described and the output MPC trajectories generated by our planner are illustrated in 2D BEV maps. Each sample consists of two plots. The left plots describe the inputs to our framework. The right plots depict the outputs of the proposed framework, which includes $X_{0:4s}^{A_{\forall k \neq AV}}$, which is the predicted positions of the road-agents in the scene, and $X_{0:4s}^{AV}$, which is the top-1 output trajectory from the planner. A trajectory is essentially a piece-wise linear path constructed from an

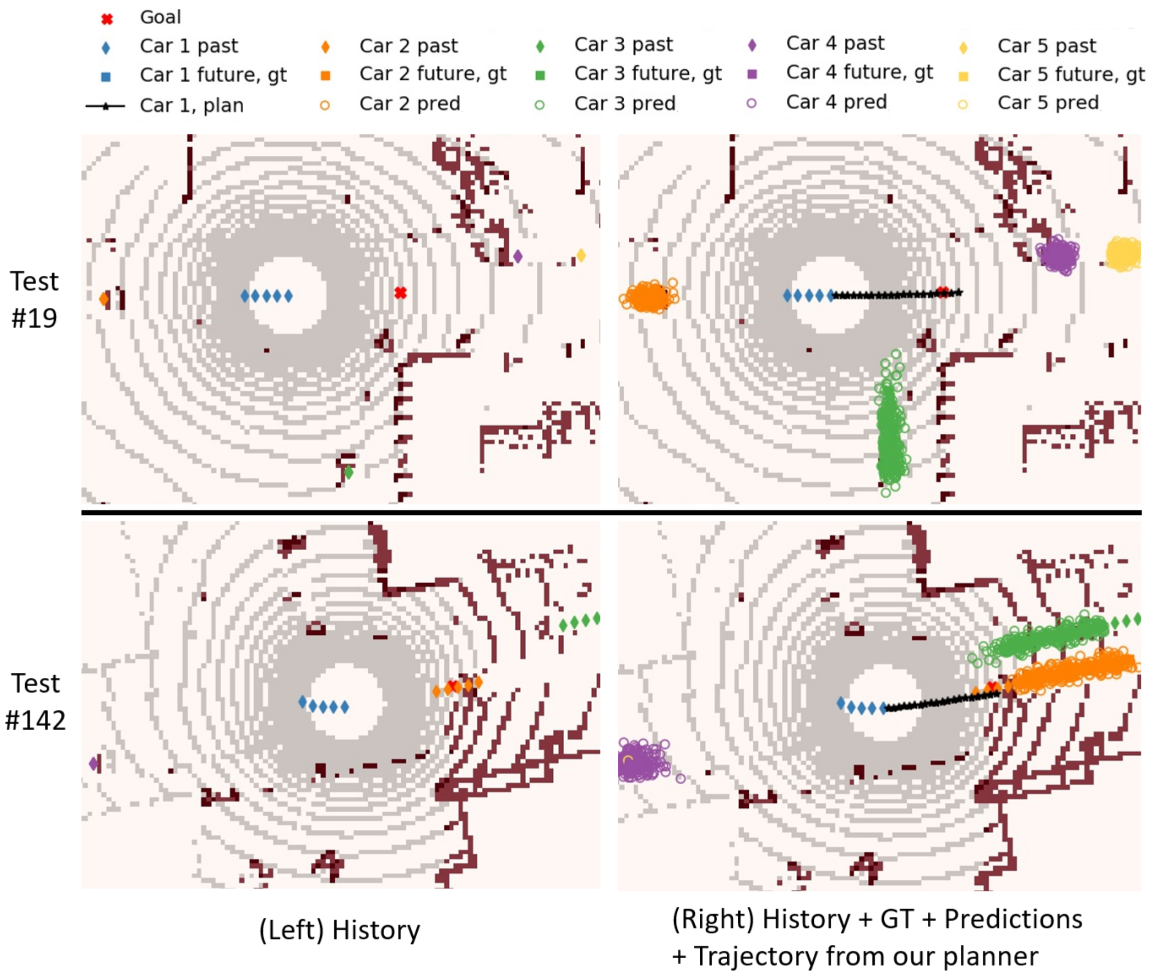


Figure 4.7: The sample environments with the goal positioned ahead of the AV.

initial state and a discrete sequence of actions $a^{AV}_{0:4s} \in R^{(20,2)}$ where each action lasts for $\Delta t = 0.2s$. Note that There exist up to 4 road-agents in the scene and we visualize 12 predictions per road-agent, $\left[X^{A_{\forall k \neq AV}}_{0:4s}\right]_{1:12}$.

Figure 4.7 depicts the simplest examples where the goal is ahead of the AV. These are arguably the easiest problem among all planning problems in our dataset since a simple policy such as the constant-speed policy would be able to find a near-optimal trajectory. In other words, no understanding of the environment such as the road topology and road-agent states is necessary. The figure suggests that the proposed planner successfully computed safe trajectories and the AV reached the goal before the end of the planning horizon.

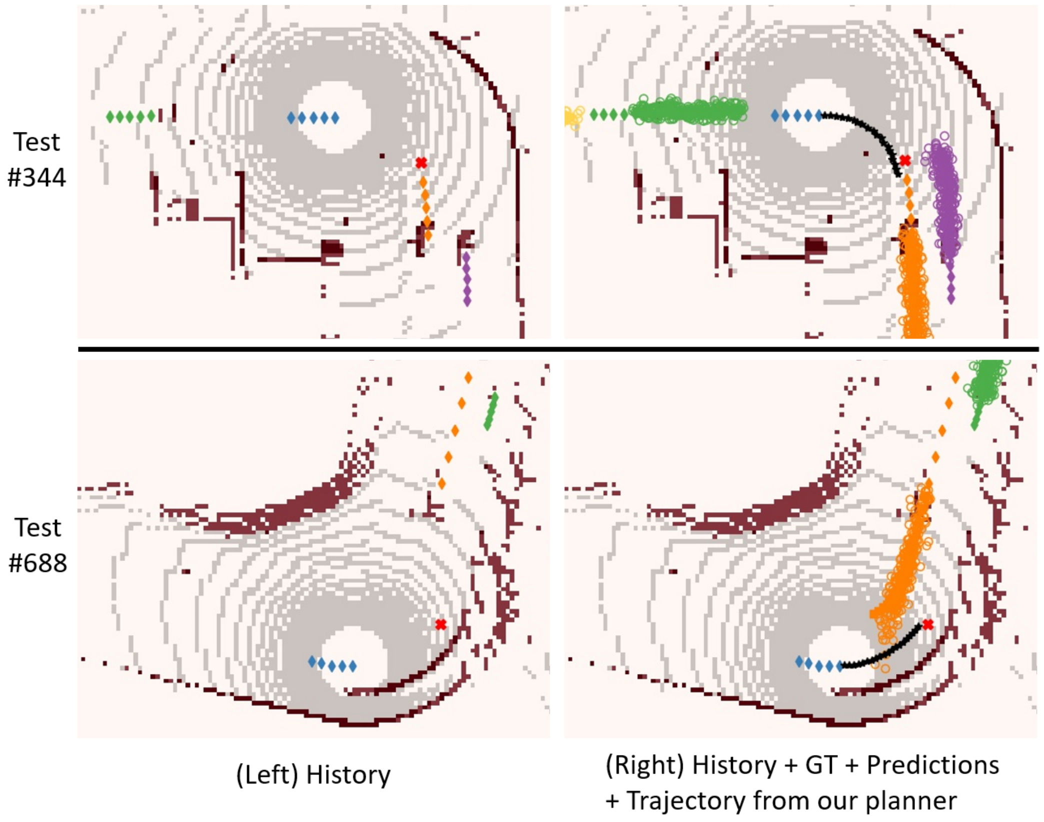

Test
#344
Test
#688
(Left) History
(Right) History + GT + Predictions
+ Trajectory from our planner

Figure 4.8: Two sample curved road environments. The goal is set to validate if the planner is aware of the spatial characteristics of the environments.

Figure 4.8 illustrates two curved road environment samples. The goal coordinates were set to check if the planner is aware of the spatial characteristics of the environment. Our planner produced action sequences that steered the AV properly. The fact that the output trajectory is curved along the road boundary suggests that the planner is spatially contextual. It is worth noting that the goal completion is determined by whether the AV passes through the vicinity of the goal, rather than the exact coordinates of the goal. We purposely designed the goal completion check in this way to encourage the planner to output trajectories flexible against uncertainties from the predictions. In addition, this makes the reward less sparse and thus, helps the training process to be faster and more stable.

The third set of examples in Figure 4.9 depicts two intersection environments with

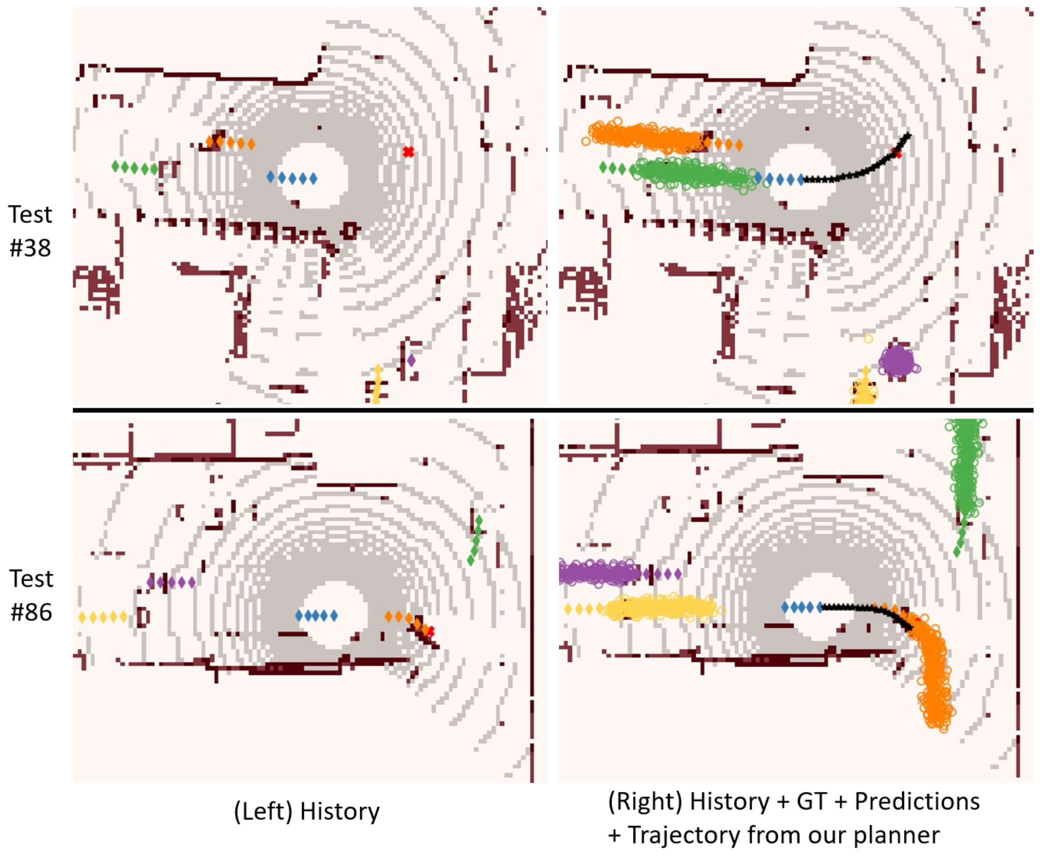


Figure 4.9: Two sample intersection environments where the goal position was set to test the planner's ability to generate left-turn & right-turn trajectories.

the goal position set to the other sides of the intersections. These examples check if the planner can produce left-turn & right-turn trajectories at intersections. For both the top (4-way intersection) and bottom sample (3-way intersection), our planner successfully outputs left-turn and right-turn trajectories. We found that the resulting action sequence incurred a small degree of jerks. This can be addressed by setting the ride comfort penalty higher or a path smoother could be used.

The last set of qualitative examples is presented in Figure 4.10. This set checks if the planner remains idle when the goal is given in the vicinity of the initial position of the AV. The figure shows that our planner was able to find that the optimal action sequence is to stay idle.

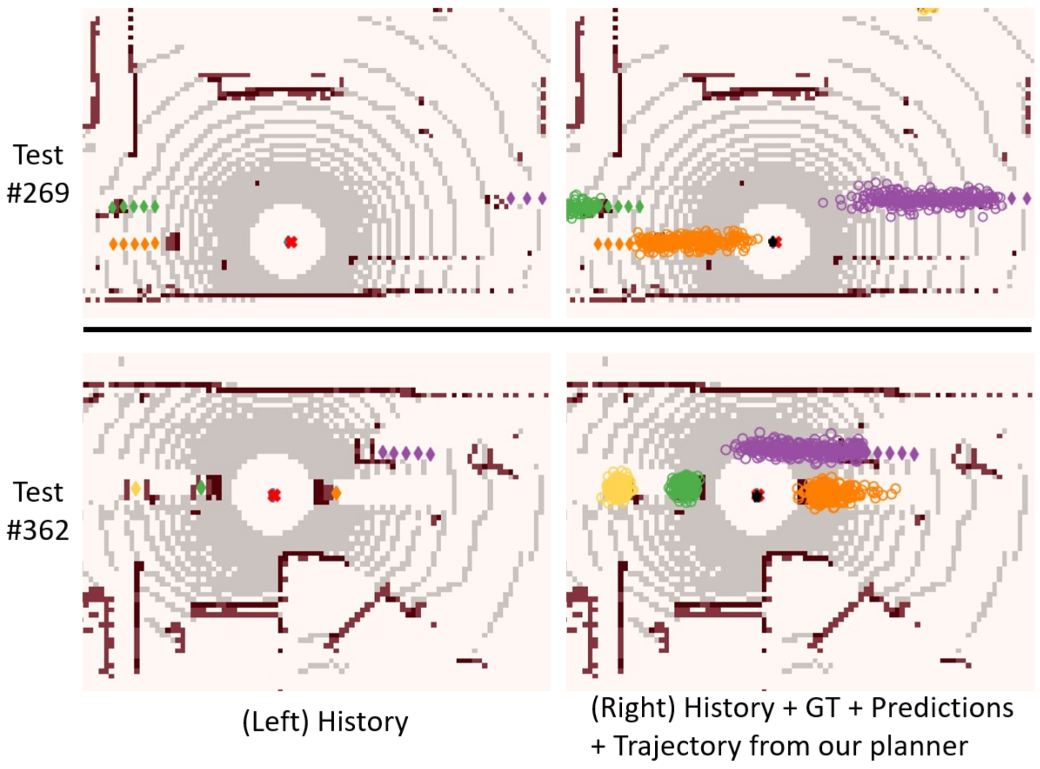

Test #269
Test #362
(Left) History
(Right) History + GT + Predictions + Trajectory from our planner

Figure 4.10: Last set of sample environments where the goal position is given in the vicinity of the initial position of the AV.

4.5.5 Quantitative Evaluation

In Section 4.5.1, we explained how we perform the quantitative evaluation. This section presents the quantitative evaluation results with a number of plots that investigate the scalabilities of the planner against the baseline planners. In other words, the quantitative evaluation checks the (1) optimality, (2) generality, and (3) computational efficiency of the planners.

First, we present the performance of all planners in terms of their average expected returns and number of failures over all 1,030 test samples in Table 4.3 and 4.4. As explained in Section 4.5.3, a failure corresponds to a negative reward, which means that the AV either (1) failed to reach the goal, (2) violated one or more traffic laws, and/or (3) collided with a road-agent.

Table 4.3 is the results obtained against $N_{cand} = 50$, which means that the prediction model (i.e., CVAE-H) generates 50 prediction samples per road-agent in the scene and the safety reward is computed against the 50 samples. Table 4.4 describes the results obtained against $N_{cand} = 100$. Higher N_{cand} indicates that the collision check is done against more trajectories of the surrounding vehicles and corresponds to higher safety penalty. By comparing the results of Table 4.3 and 4.4, we can infer sensitivities of the safety of the planners against the number of prediction samples.

Table 4.3: Average optimality and number of failures of the planners over 1,030 test samples, with 50 prediction samples per road-agent.

Method	**Mean Expected Returns**	**Failures**
Dynamic programming	89.7	**0**
Proposed planner (1,024 samples)	**91.7**	**0**
Naive random shooting (10,240 samples)	69.4	**0**
Naive random shooting (1M samples)	73.7	**0**
Behavior cloning (1,024 samples)	81.4	12
Behavior cloning (10,240 samples)	84.2	2
Rule-based 1: Constant speed $a^{AV} = 0(m/s^2)$	-37.1	605
Rule-based 2: Constant accel $a^{AV} = 0.5(m/s^2)$	-106.8	646
Rule-based 3: Constant brake $a^{AV} = -0.5(m/s^2)$	-25.9	850
Heuristics 1: Modified IDM	-75.8	649
Heuristics 2: Goal-conditioned acc	85.9	41

Table 4.3 reports the average expected returns and number of failures of the planners against $N_{cand} = 50$. Figure 4.11 normalizes the results presented in Table 4.3 against the average DP optimality (i.e., the global optimality). We draw a number of observations about the results in the following paragraphs.

Dynamic Programming. Dynamic Programming achieved high a mean expected return and zero failure. This is anticipated as it produces the global optimal solution with respect to the *resolution of the state-action space*. As we discussed in Section 4.5.2, we use DP only for the benchmarking purpose as DP takes a minute to produce a

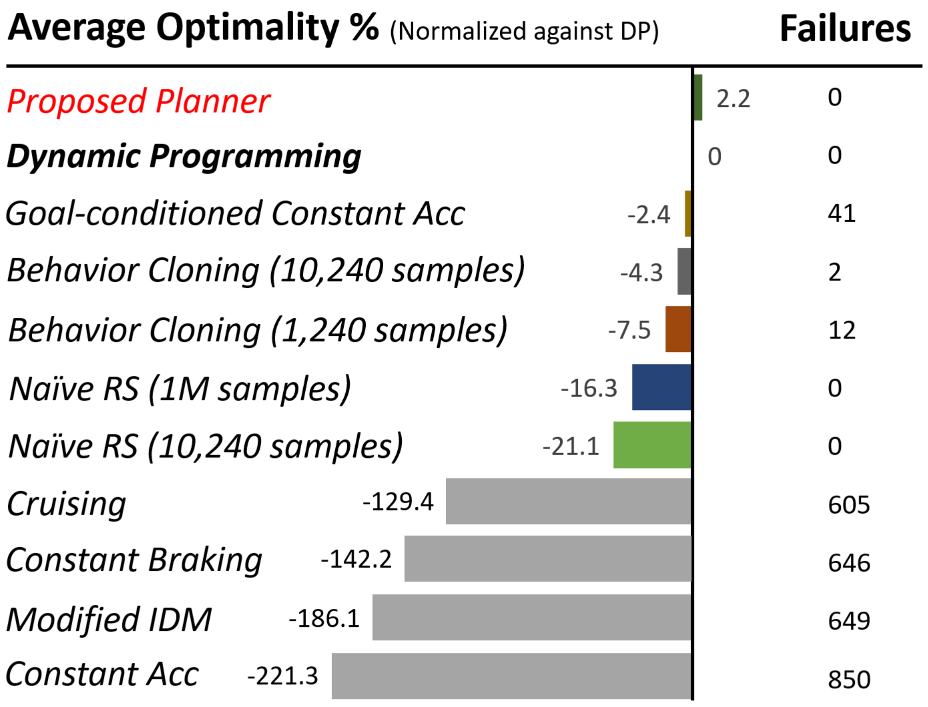


Figure 4.11: The average optimalities normalized against the DP and the numbers of failures out of the test set of size 1,030, as presented in Table 4.3.

single trajectory. As aforementioned, the DP solutions were computed using a coarse state-action space with a low planning frequency. Due to the coarse resolution of action and state spaces, the DP solution incurred higher ride comfort penalties on average and thus scored slightly lower than the proposed planner.

Rule-based Models. All of the rule-based approaches achieved low optimality scores and failed in most of the test environments. This happened because they either failed to reach the goal, collided with a road-agent, and/or violated the speed limit in most of the test examples.

Heuristics 1: Modified IDM. The first heuristic approach, the modified IDM, resulted in low expected returns and failed more than 649 out of 1,030 test examples. This is because a lot of test examples involve some degrees of steering (i.e., their goal positions are off the x-axis) and do not have a preceding vehicle to the AV. While IDM performed moderately well in the specific car-following scenarios where the goal positions were set along the travel direction of the preceding vehicles, IDM failed to reach the goal in all the other scenarios.

Heuristics 2: Goal-conditioned acceleration. The goal-conditioned acceleration planning is arguably the most powerful baseline. It resulted in the highest average expected returns among all baselines (less dynamic programming). As we designed this heuristic to always reach the goal, abide by the traffic rules, and maximize ride comfort, the resulting trajectory only incurs a small acceleration penalty. However, it does not consider the positions of the road-agents in the scene, which is safety-critical. As a consequence, this baseline led the AV to collide into a road-agent and failed 41 test examples.

Naive random shooting. The performance of naive random shooting depends on N_{plan}, the number of candidate trajectory plans. As we described in Section 4.4.3, this is true for all shooting planners that select the top-1 trajectory among all candidate trajectories. Although a higher N_{plan} corresponds to a higher optimality, it also means higher computational time and memory requirement. To quantify the impact of N_{plan}, we report the results for $N_{plan} \in \{10240, 1024000\}$. We find that it takes on average 0.7s and 290MB to run the naive random shooting with 1,024,000 (1M) candidate trajectories. Although the naive random shooting planner achieved decent optimalities, we argue that $N_{plan} > 1024000$(1M) is not suitable for real-time use.

Behavior cloning. Similar to Naive random shooting, the performance of BC policy depends on the number of samples. This is because the backbone of BC is a conditional VAE distribution, which generates stochastic outputs. The fact that the dataset that BC policy was trained on is multi-modal contributes to the performance dependency on the number of samples. We used $N_{plan} \in \{1024, 10240\}$ and observed that the BC policy performs moderately well across various scenarios, however, it failed in a few environments due to the distribution drift problem [147]. We have not investigates the number of samples over 10,240 for the computational efficiency reason that is discussed later in this section.

The proposed planner. The proposed planner performs the best among all the baselines in terms of the expected returns and number of failures. While the resulting trajectory and action sequences involved some degrees of jerks, the proposed planner had zero failure; this means that it always reached the goal, never collided with a road-agent in the scene, nor violated a traffic law. Furthermore, it produced near-optimal solutions whose average expected return is close to 100, and even higher than the DP. Later in this Section, we show that the proposed planner is not only effective, but also general and computationally efficient. Similar to Naive random shooting and behavior cloning methods, the proposed planner is a sampling based method and we used 1,024 samples to compute the top-1 trajectory plan.

Table 4.4: Average optimality and number of failures of the planners, with 100 prediction samples.

Method	**Mean Expected Returns**	**Failures**
Dynamic Programming	89.6	**0**
Proposed Planner (1,024 samples)	**91.6**	**0**
Naive random shooting (10,240 samples)	69.4	**0**
Naive random shooting (1M samples)	73.7	**0**
Behavior cloning (1,024 samples)	80.6	21
Behavior cloning (10,240 samples)	83.6	5
Rule-based 1: Constant speed $a^{AV} = 0 (m/s^2)$	-48.3	610
Rule-based 2: Constant accel $a^{AV} = 0.5 (m/s^2)$	-117.0	653
Rule-based 3: Constant brake $a^{AV} = -0.5 (m/s^2)$	-37.8	853
Heuristics 1: Modified IDM	-99.6	659
Heuristics 2: Goal-conditioned acc	79.4	44

On the other hand, Table 4.4 reports the results with $N_{cand} = 100$. As aforementioned, higher N_{cand} means higher safety risk. In comparison with the results of Table 4.3, we see that rule-based and heuristic-based methods all have lower expected returns and higher numbers of failures with the increase in the number of prediction samples (i.e., $N_{cand} = 50$ vs 100). In contrast, N_{cand} has minimal impact on the proposed and naive random shooting methods. The rule-base and heuristic-based methods output deterministic trajectories and thus, scale worse than the proposed method with respect to the number of trajectories of the road-agents in the scene.

Table 4.5: Average computational requirements of the planners.

Method	**Computation Time (GPU)**	**Memory Consumption**
Dynamic Programming	60s	3.0GB
Proposed Planner (1,024 samples)	0.009s+0.03s	0.7GB
Naive RS (10,240 samples)	0.02s+0.03s	0.03GB
Naive RS (1M samples)	0.7s+0.03s	0.4GB
Behavior cloning (1,024 samples)	0.12s+0.03s	0.9GB
Behavior cloning (10,240 samples)	0.61s+0.03s	2.0GB
Rule-based 1: Constant Speed (Cruising)	<0.01s	<0.01GB
Rule-based 2: Constant Acceleration	<0.01s	<0.01GB
Rule-based 3: Constant Braking	<0.01s	<0.01GB
Heuristics 1: Modified IDM	0.01s	0.02GB
Heuristics 2: Goal-conditioned C.Acc	<0.01s	<0.01GB

Second, we report the results of the analysis on the computational requirements for the baselines and proposed planner in Table 4.5. We use two metrics; computation (inference) time and memory consumption. We compute the inference time by measuring the time to run a planner and obtain the output trajectory. Memory consumption is simply measured by the amount of the memory occupied by the planner. All results are measured with GPU as opposed to CPU to allow the planner to leverage parallel computation techniques.

The computation time that is presented in Table 4.5 is divided into the two separate times by the $'+'$ sign; (1) the time to produce N_{plan} candidate trajectories and (2) the time to compute the top-1 trajectory among the candidates. While the latter is more or less the same for all sampling-based planners, the former changes drastically depending on the planner. The computation time was measured using a single Nvidia GeForce RTX 2070 SUPER (8GB) GPU on Ubuntu. The memory consumption primarily comes from the parameters of the backbone neural networks of the planners and the tensors that store candidate trajectories.

As presented in Table 4.5, the proposed planner is much faster than all the other sampling-based methods. To be specific, it takes less than 0.01s to produce candidate trajectories and 0.03s to evaluate the candidates and select the top-1 trajectory. Compared to the behavior cloning or naive RS planners, the sampling process (i.e., generation of candidate trajectories) is orders of magnitudes faster. For the behavior cloning policy, we found that $N_{plan} > 1024$ is not computationally efficient enough for the planning task. In terms of memory consumption, our planner takes roughly 0.7GB to store the backbone neural network. Considering that a modern personal GPU memory is around 6-12GB and that an autonomous vehicle typically has one or more GPUs with higher computation powers, we argue that our planner is computationally efficient.

Third, we investigate the generalities of the baselines and proposed planner using a box plot of their optimalities over all 1,030 test samples. In a box plot, the band inside the box represents the median. The top and bottom edges of the box are the first and third quartiles. The ends of the whisker extend to the extreme data that are not considered outliers, and the outliers are indicated by a 'o'.

Figure 4.12 displays the resulting box plot. A generalizable planner should perform near-optimal across diverse environments. The box plot metrics including the minimum, 25% quartile, and median are indicators of the generality of the planners. The result shows that our metrics outperform all the other planners. The proposed planner achieves near-optimality in the majority of the environments, except a few environments where the output trajectories did not favor ride comfort or travel time much. One noticeable result is that our planner did not fail a single test example.

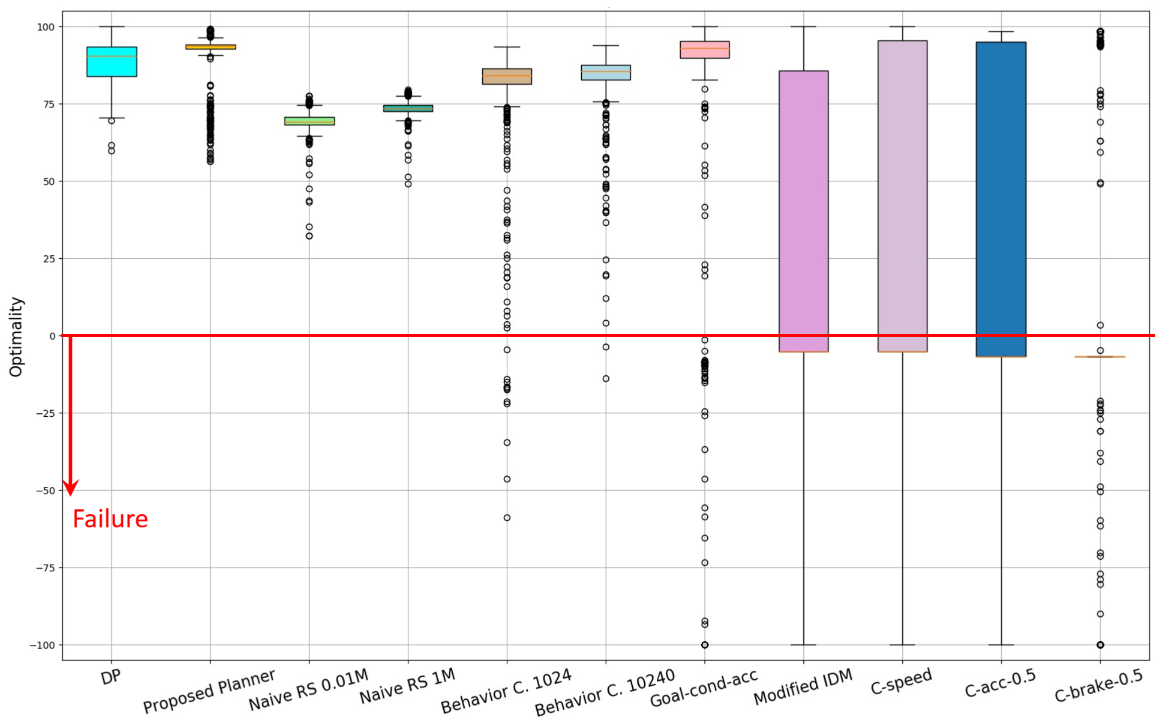


Figure 4.12: The box plot of the expected returns over all 1,030 test samples.

While the goal-conditioned constant acceleration policy has good scores for many box plot metrics, there exist many outliers with negative expected returns. In short, the proposed method is generalizable.

We share the violin version of the box plot in Figure 4.13. A violin plot is similar to a box plot in that it includes all the markers and ranges of the box plot, except that it provides the probability density of the data additionally. The provided probability density is smoothed by a kernel density estimator. The resulting violin plot agrees with the results presented in the box plot.

Based on the experiment results, we place the planning methods we tested into the quadrant depicted in Figure 4.14. The x-axis indicates the computational efficiency and the y-axis denote the optimality & generality.

In this section, we shared the evaluation results of the proposed planning approach using both qualitative and quantitative metrics. Our evaluation paid special attention

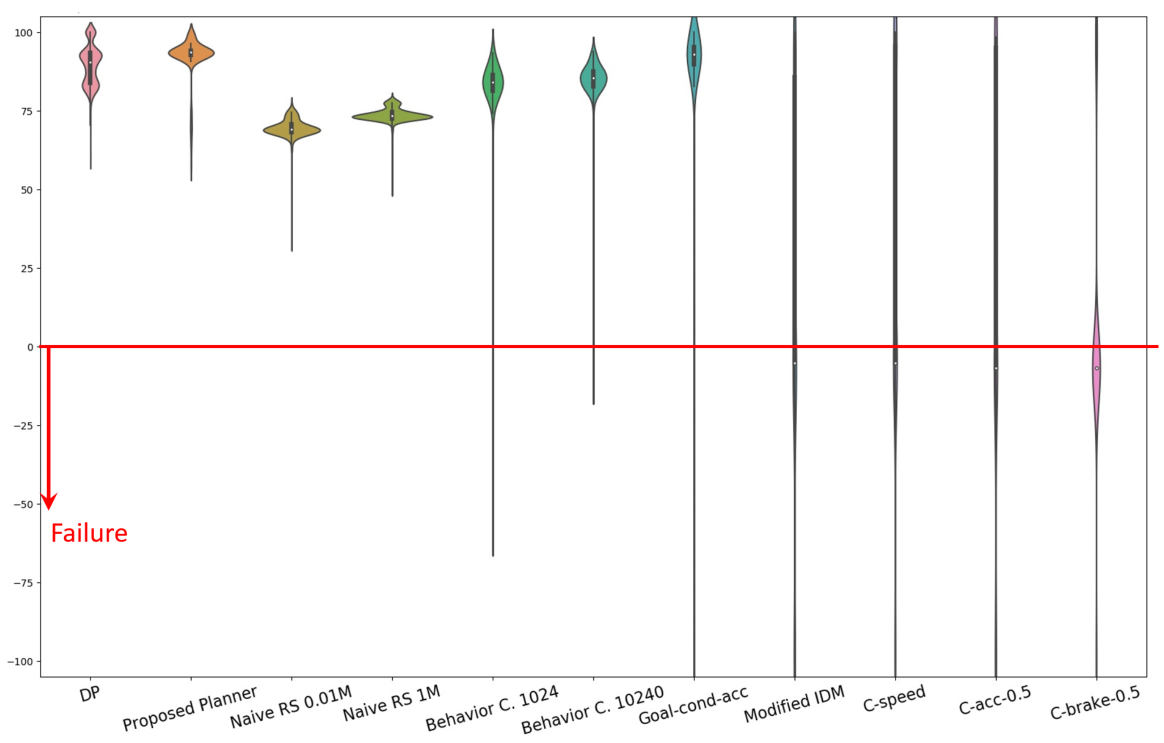


Figure 4.13: The violin plot of the expected returns over all 1,030 test samples.

to the scalability of the planner and reported that the proposed planner is near-optimal, general, and computationally efficient. To this end, we conclude that *the proposed planner is scalable and thus serves as the ideal planner for our framework.*

4.6 Trajectory vs Probabilistic Occupancy Map

Recall that the contribution of the thesis is three-fold; the two novel prediction models detailed in Chapter 3, the novel planner described so far in this Chapter, and lastly the framework that leverages probabilistic occupancy map as the uncertainty (prediction) representation. We dedicate the last section of Chapter 4 to the third contribution. In this sense, we re-perform the planning task with the POM representation this time, evaluate its scalability, and identify its advantages over the trajectory representations.

As we elaborated in Section 2.3 and 4.4.2, in theory, the POM representation

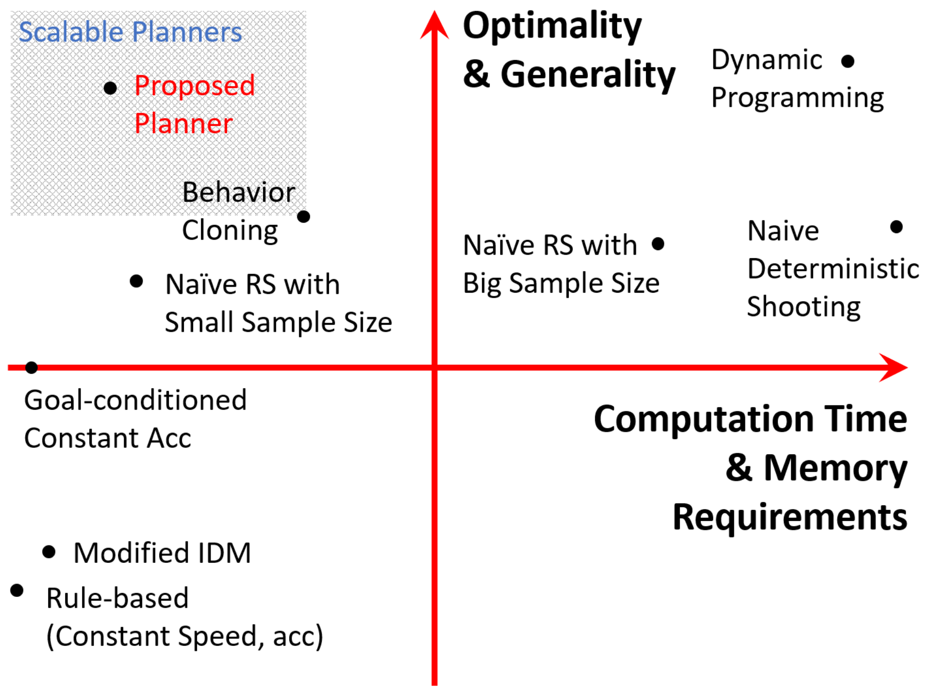

Scalable Planners
Proposed Planner
Behavior Cloning
Naïve RS with Small Sample Size
Optimality & Generality
Dynamic Programming
Naïve RS with Big Sample Size
Naive Deterministic Shooting
Goal-conditioned Constant Acc
Computation Time & Memory Requirements
Modified IDM
Rule-based (Constant Speed, acc)

Figure 4.14: We place the tested planning methods into a quadrant of performance and computational efficiency (i.e., the keywords of the scalable planner).

allows the planner to compute the solutions in a shorter time compared to the trajectory representation. In particular, the POM representation has better computational scalability with respect to the complexity of the environments (e.g., number of road-agents in the scene). To examine if the theoretical advantage carries over into the experiments, we investigate the computation time and memory requirements for our prediction-planning framework with the POM representation.

For the fair comparison, we performed the same experiment presented in Section 4.5. Recall that our prediction-planning framework works the same for the POM representation. This means that we used the same implementation of the framework as we aim to minimize biases in pre-processing, post-processing, and performance evaluation. The only difference is that the POM representation utilizes a HCNAF model in place of a CVAE-H model as described in Figure 1.4, 3.1, and 4.4. Another difference is that the safety reward is computed using Equation 4.5 instead of 4.4. In this regard, the difference in the computation time is explained by the difference in the run-time of HCNAF versus CVAE-H and the safety reward computations. Similarly, the memory requirement difference originates from the dissimilar memory requirements of the two prediction models and the safety reward computations.

In order to minimize the computation time, we fully parallelized our implementation of the framework (e.g., pre-processing pipelines, prediction inference, planning, and so on). This is done by minimizing iterations and loops and maximizing the use of the batch computing of PyTorch [158]. For example, the time dimension, prediction sample dimension, and candidate trajectory plan dimension are all cast onto the batch dimension. While this increases memory usage, it provides a significant speed-up for tensor computations.

Table 4.6 - 4.8 deliver the resulting computational efficiency measurements of the two approaches. The two most important parameters are N_{plan} and N_{pred}, which represent the number of candidate trajectory plans sampled from the planner and the total number of trajectory prediction samples from the prediction model (i.e., CVAE-H). Based on Equation 4.4, N_{pred} can be computed as follows.

$$N_{pred} = N_{cand} N_T K, \tag{4.11}$$

where N_{cand}, N_T, and K each indicates the number of trajectory prediction samples per road-agent in the scene, number of planning steps, and number of road-agents in the scene. In our experiment, we set $N_T = 20$ and $K = 4$. N_{cand} is a tunable hyper-parameter that determines the degree of safety conservatism (see Section 3.10 and Equation 4.4 for the detail). For example, $N_{cand} = 100$ (i.e., the number of prediction samples used in Table 4.4) corresponds to $N_{pred} = 8000$.

We inform that the reported computation time was measured end-to-end (i.e., a complete cycle of the framework described in Figure 1.4). This includes pre-processing, prediction, planning, and post-processing. The memory consumption indicates the memory size required to run the framework end-to-end.

Table 4.6: Trajectory vs POM - a comparison of their computational efficiencies.

Method	**Computation Time (GPU)**	**Memory Consumption**
Predict & Plan with *Trajectory* (CVAE-H) N_{plan}=128, N_{pred}=8,000	0.088s	1.5GB
Predict & Plan with *POM* (HCNAF) N_{plan}=128	0.081s	0.4GB

Table 4.6 shows that the POM representation is more computationally efficient under the nominal hyper-parameters (i.e., N_{plan}=128, N_{pred}=8,000). When POM is used as the prediction and uncertainty representation, the run-time of our framework is roughly 17% is faster compared to our trajectory-based prediction & planning. Furthermore, the memory consumption is significantly less. Considering that the prediction models CVAE-H and HCNAF have a similar number of network parameters around 10 million, the memory consumption comes from the architectural difference and computational scalability of N_{pred}.

A planner that leverages POM as the prediction representation does not require a trajectory sample to compute the safety reward. Conversely, a planner with the trajectory representation requires $N_{plan} \cdot N_{pred}$ number of safety checks to compute the safety reward. Instead of iterating through the safety check (Equation 4.4) $N_{plan} \cdot N_{pred}$ times, we fully parallelize the computation. As a result, the check is only performed once, however, it increases the memory requirement to store the increased batch.

We emphasize that the numbers reported in 4.6 are tied to the hyper-parameters N_{plan}=128, N_{pred}=8,000. It is because that the computational efficiency is largely affected by the hyper-parameters. This is closely related to how the safety rewards are computed for the trajectory representation (Equation 4.4) and the POM representation (Equation 4.5). In the next two tables, we analyze the impact of the two hyper-parameters on computational efficiency.

Table 4.7 investigates the computational efficiency with respect to N_{pred}, the total

Table 4.7: The computational efficiency vs number of trajectory samples N_{pred}.

Method	**Computation Time (GPU)**	**Memory Consumption**
Predict & Plan with *Trajectory* (CVAE-H)		
N_{plan}=128, N_{pred}=960	0.076s	1.0GB
N_{plan}=128, N_{pred}=8,000	0.088s	1.5GB
N_{plan}=128, N_{pred}=80,000	0.113s	3.5GB
Predict & Plan with *POM* (HCNAF)		
N_{plan}=128, N_{pred}=0	0.081s	0.4GB
N_{plan}=128, N_{pred}=inf	0.081s	0.4GB

number of trajectory prediction samples for the road-agents in the environment. Since the POM-based planning does not require a trajectory sample, N_{pred} has zero impact. However, for the trajectory-based implementation, both the computation time and memory requirement increase with respect to N_{pred}. Since N_{pred} can quickly grow as the number of road-agents, planning horizon, and resolution increase, this may limit the computational scalability of the trajectory-based planning especially when N_{cand} is not small.

Table 4.8: The computational efficiency vs number of candidate plans N_{plan}.

Method	**Computation Time (GPU)**	**Memory Consumption**
Predict & Plan with *Trajectory* (CVAE-H)		
N_{plan}=16, N_{pred}=8,000	0.081s	1.5GB
N_{plan}=128, N_{pred}=8,000	0.088s	1.5GB
N_{plan}=1,024, N_{pred}=8,000	0.090s	1.6GB
N_{plan}=10,240, N_{pred}=8,000	0.113s	3.2GB
Predict & Plan with *POM* (HCNAF)		
N_{plan}=16	0.068s	0.2GB
N_{plan}=128	0.081s	0.4GB
N_{plan}=1,024	0.173s	3.0GB

Table 4.8 describes how the computational efficiencies of the two implementations change with respect to N_{plan}. First, the impact of N_{plan} on the trajectory-based prediction and planning is negligible as the metrics remain similar for all three different

numbers of N_{plan}. While N_{plan} theoretically does affect the number of safety checks, we found that the influence of N_{plan} is small when it is under a certain threshold. This is because N_{plan} only changes the number of the size of planning-related tensors whereas N_{pred} affects the forward-propagation of the CVAE-H model to obtain prediction samples and evaluate their uncertainties. In addition, N_{pred} increases the computational load of all downstream tasks of the prediction module.

Unlike the trajectory-based implementation, the POM-based implementation computational efficiency is not affected by N_{pred}, as shown in Table 4.7. However, both computational time and memory requirement quickly surge with respect to N_{plan}. This is due to the computation of the safety rewards involving N_{plan} number of probabilistic occupancy evaluations (i.e., forward propagation of the HCNAF model) as explained in Equation 4.5. In short, the POM-based implementation has limited computational scalability with respect to the number of total candidate trajectories.

We summarize the comparisons between the trajectory-based and POM-based implementations presented in Table 4.6 - 4.8 in the following three bullet points.

1. First, the trajectory-based implementation computationally scales poorly with respect to the number of road-agents in the scene and the number of the prediction samples. This may limit the application of the trajectory-based implementation in complex environments with large numbers of road-agents that can create diverse and multi-modal uncertainties in the future.

2. Second, the POM-based implementation scales better with respect to the complexity and stochasticity of the environments as POM is independent of the number of road-agents.

3. Third, compared to the POM-based implementation, the trajectory-based implementation is more computationally efficient for the planners that sample a high number of candidate trajectory plans.

We conclude that the POM-based prediction & planning has higher computational efficiency compared to the trajectory-based prediction & planning under the *nominal* hyper-parameters. Depending on the complexity of the environment, degree of the uncertainties in the environment, and importance of N_{plan}, one could make an informed decision for using either trajectory or POM as the prediction & uncertainty representation. Since the two hyper-parameters N_{plan} and N_{pred} influence the qualities of both prediction outputs and trajectory plans, one should make the decision carefully in consideration of the capacities of the prediction model and planner.

CHAPTER V

Conclusion

5.1 Summary of The Thesis

This dissertation proposes a scalable prediction and planning approach for autonomous driving in urban driving environments. My approach consists of two main tasks: (1) probabilistic prediction of the environments and (2) uncertainty-aware and contextual trajectory planning for AV. A framework that bridges both prediction and planning tasks is proposed and is implemented using two different prediction representations; (1) trajectory-based and (2) probabilistic occupancy map-based representations. The thesis first investigates the performances of the proposed prediction module separately and later examines the scalability of the framework with both of the two prediction representations.

I started the thesis by describing the challenges of self-driving in urban areas in Chapter 1. I explained how scalability is critical to the success of autonomous driving and defined the three keywords of the scalability: effectiveness, generalizability, and computational efficiency. I explained that data-driven (i.e., learning-based) approaches are great tools for a scalable solution to the problem. Accordingly, a framework that includes deep learning prediction and planning modules was proposed.

Chapter 2 was dedicated to the mathematical formulations of the problem and solution implementations. Specifically, I argued that the non-autoregressive implementation for prediction and planning is more suitable as it is more computationally efficient compared to the autoregressive implementation. I explained how computation resources may scale with respect to the two implementations. Furthermore, I analyzed the two possible prediction representations (trajectory and POM) and compared their advantages. Lastly, I explored a number of publicly available datasets to select the best-suited dataset for the research.

In Chapter 3, I first defined the four attributes of the ideal prediction model for autonomous driving in urban areas. I then reviewed existing approaches to the prediction task as well as the preliminary deep learning concepts of the proposed models. In the middle of the Chapter, I introduced two forecasting algorithms, namely HCNAF and CVAE-H, that are conditional generative models which approximate complex probability density functions and generate predictions as well as the uncertainties associated with each prediction. I elaborated on their mathematical and architectural details and how the models are adapted for the prediction task with two different prediction representations; trajectories and POM. Lastly, I conducted experiments to evaluate the proposed prediction models in terms of the four attributes of the ideal prediction model and show that they effectively leverage spatio-temporal sensor information and predict the trajectories of the road agents in diverse environments.

In Chapter 4, the planning task was depicted. I started the chapter by introducing the objective and goal of the planning task. I then reviewed a number of planning approaches that are popularly used in the field of robotics and autonomous driving. I analyzed the advantages and disadvantages of each method and used them as rationales to design an optimal, generalizable, and computationally efficient planner. I proposed a planner that autonomously produce action sequences near-optimal to the state of the environment and design such actions to be aware of the uncertainties captured in the prediction outputs. The proposed planner is a sampling-based random shooting planner based on a Gaussian mixture distribution. Experiments confirm that the proposed planner generates safe and contextual trajectories under various environments in real-time, and the performance compares favorably against the dynamic programming results.

Later in Chapter 4, I compared the computational efficiencies of the two solution implementations with the two uncertainty representations. I examined their computational efficiencies by running the trajectory-based and POM-based implementations end-to-end. The results show that under the nominal condition, the probabilistic occupancy map representation leads to more computationally efficient trajectory generations as it removes the need for an excessive sampling of future states of the road-agents. In addition, I analyzed how the two representations computationally scale with respect to the two hyper-parameters that determine the qualities of the prediction and planning.

5.2 Future Works

Better Backbone Network for The Prediction Models

The proposed prediction models, HCNAF and CVAE-H, both leverage hyper-networks which are the backbone of the prediction models. The hyper-networks are deep neural networks that extract social and spatial information about environments and provide the encoding of the environments to the main networks so that they can make contextual predictions. In this regard, the design of the hyper-network greatly affects the performance of our prediction models. In this thesis, we constructed the hyper-network using MLP, CNN, RNN, and simple attention models as depicted in Figure 3.9. However, one could improve the hyper-network architecture by leveraging other deep learning networks.

Transformer [71] is one example that could better extract social features from the environments. Transformer builds on multiple stacks of different attention layers and outperforms RNN and simple attention-based models. Known as one of the most successful deep learning models, Transformer has been used across different fields of artificial intelligence and machine learning such as NLP [159, 160] and Speech [161]. As the autonomous driving prediction problem could be formulated as a sequence prediction problem, Transformer can help better understand the social contexts of driving. As a matter of fact, we see Transformer-based prediction models [162] being used in the literature recently. We expect the utility of Transformer to grow with respect to the complexity of the environment such as the number of road-agents in the scene. This makes Transformer a great candidate for the urban driving prediction problem.

Another example is graph-based models or Graph Neural Network (GNN). GNN is a type of deep learning model designed to work with data represented using graphs. Defined as a set of vertex and edges (i.e., $G = (V, E)$), the graph can model complex relationships and inter-dependencies between vertices. The graph-based models offer a powerful way of performing graph-level inferences (macroscopic) as well as node-level or edge-level inferences (microscopic). For autonomous driving, a graph could represent the complex interactions between multiple objects in the scene where the node denotes a road-agent and the edge indicates the interaction between two road-agents. A number of recent publications have applied this graph-based approach to self-driving prediction problems and showed promising performances [163, 164]. In this regard, an interesting extension of our work would be adding graph-based models to our prediction model to better extract social features of urban driving environments.

Extracting Rewards via Inverse Reinforcement Learning

In section 4.1, we introduced the motivation behind designing the reward function for training and evaluating the proposed planner. In section 4.4.2, we presented the mathematical formulation of the reward function. In short, to construct the reward function, we used the common knowledge about transportation and our intuitions about driving. While this approach is intuitive and interpretable, we may have failed to capture other aspects of driving into the reward. Even if we captured the essential features of the decision-making, we often do not know the optimal weight ratio between the reward terms. This could be critical to learning-based planners as the trajectories produced by the learning-based planners differ significantly depending on the weights between reward terms. This means that it is inevitable to go through a tedious tuning process to search for a working set of weights.

We could instead utilize human driving data to infer what constitutes the optimal, or at least reasonable and safe, decision. This approach is known as Inverse Reinforcement Learning (IRL) which is the problem of inferring the reward function of an intelligent agent, often denoted as an expert, by observing its behavior. A number of popular methods exist including maximum-entropy IRL [165] and bayesian IRL [166]. The application of IRL to autonomous driving is not a trivial task as it requires a careful design of the reward approximator neural network, integration to the planner, and consideration of uncertainties in the environments are necessary. For this reason, IRL has been an active research area for self-driving [13].

Expanded Planning Objective: Energy-efficiency

Our reward design, which was used for planning, considers several aspects of transportation such as travel time, safety, and ride comfort. We could expand the reward to include another important aspect of transportation; energy efficiency. One of the biggest sources of travel cost is energy consumption. Indeed, traffic congestion in urban driving environments costs Americans an extra \$160 billion worth of fuel annually [103]. On top of the fuel-saving, energy-efficient driving has environmental and ecological benefits as it reduces CO2 emissions [167].

In order to make the AV be energy-efficient and environment-friendly, we would design an energy consumption reward term, r_{energy}, and include it as a part of the planning reward. The energy efficiency of a vehicle depends on various factors including the use of throttles, brakes, weather, and the vehicle types (either fuel-based vehicle or electric vehicle). Factors that autonomous driving agents can control include acceleration, steering, and route choices. A number of studies have shown that the speed and acceleration profile have significant impacts on fuel consumption [112, 168] as it is directly related to operations of engines and motors. In this regard, r_{energy} would be a function of acceleration, speed, and travel time. In this sense, energy efficiency would be a useful addition to the reward function and help reduce the operation cost of the AV.

Other Planning Approaches

Our planner can be classified as a model-based RL algorithm and is closely related to the policy gradient method. This is because we use a Gaussian mixture as the policy, train the policy using the expected returns, and perform inference by sampling the policy. The difference from typical policy gradient methods is the objective function that includes the cross-entropy term (see Section 4.4.6).

Considering the similarities between policy-based RL methods and ours, we could instead use a model-based policy gradient model to solve the planning task. Although policy gradient methods are less interpretable than our method, policy-based RL methods are as scalable as our approach. Moreover, recent policy gradient methods have successfully solved various complex robotics and control tasks [139, 144, 11, 145, 169].

Imitation learning is another approach we could further investigate. Recall that we have used a behavior cloning planner as one of our baselines. As discussed in Section 4.2.6, the behavior cloning method has some limitations. There have been a number of approaches that addressed the limitations. One of the approaches we could potentially explore is Plato [148] which leverages an optimal control algorithm to label roll-outs from the behavior cloning policy. If we were to apply Plato to our planning problem, we could use our behavior cloning policy as the initial policy and iteratively update the policy by leveraging the dynamic programming policy as the labeler.

Extension of Our Approach to Other Datasets

In this thesis, we used the PRECOG-Carla dataset [25] for the evaluation of the proposed prediction and planning models. As we discussed in Section 2.4, we intentionally chose it since it provides cleaner and well-processed sensor information and labels compared to other datasets that are typically more noisy and missing data. Moreover, the PRECOG-Carla dataset consists of generic urban driving environments and has a number of benchmarks available to the public. While these advantages allow us to minimize data cleansing efforts and focus on the designs of prediction and planning models, the PRECOG-Carla dataset is certainly not the most realistic nor complex dataset.

In fact, there exist other large-scale datasets such as Argoverse [20], Lyft Level 5 [21], and NuScenes [22]. Similar to the PRECOG-Carla dataset, they all include diverse urban driving environments with vision sensor information and trajectory labels and thus our framework can directly be applied to these datasets. As the datasets were established using real-world driving examples in urban areas, the datasets are relatively more complex and realistic.

In this sense, we would like to extend our work to one or more of these datasets. This means that additional data cleansing processes, data imputation, and/or post-processing are necessary as they have significant impacts on the performance of the prediction and planning models. Since each dataset uses a different suite of sensors and evaluation metrics, the network architectures would be customized accordingly to the available sensors such as camera and radar.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2446–2454, 2020.

[2] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[3] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA urban challenge: autonomous vehicles in city traffic*, volume 56. springer, 2009.

[4] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE intelligent vehicles symposium (IV)*, pages 163–168. IEEE, 2011.

[5] Geunseob Oh and Jean-Sebastien Valois. Hcnaf: Hyper-conditioned neural autoregressive flow and its application for probabilistic occupancy map forecasting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14550–14559, 2020.

[6] Geunseob Oh and Huei Peng. Cvae-h: Conditionalizing variational autoencoders via hypernetworks and trajectory forecasting for autonomous driving. *arXiv preprint arXiv:2201.09874*, 2022.

[7] Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

[8] Edward Schmerling, Karen Leung, Wolf Vollprecht, and Marco Pavone. Multimodal probabilistic model-based planning for human-robot interaction. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[9] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[10] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in Neural Information Processing Systems*, 31:4754–4765, 2018.

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

[12] Paavo Parmas, Carl Edward Rasmussen, Jan Peters, and Kenji Doya. Pipps: Flexible model-based policy search robust to the curse of chaos. In *International Conference on Machine Learning*, pages 4065–4074. PMLR, 2018.

[13] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[14] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[15] Yi Xiao, Felipe Codevilla, Akhil Gurram, Onay Urfalioglu, and Antonio M López. Multimodal end-to-end autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[16] Aditya Prakash, Kashyap Chitta, and Andreas 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.

[17] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[18] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE access*, 8:58443–58469, 2020.

[19] Anusha Nagabandi, Kurt Konolige, Sergey Levine, and Vikash Kumar. Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112. PMLR, 2020.

[20] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, et al. Argoverse: 3d tracking and forecasting with rich maps. In *Proceedings*

of the IEEE Conference on Computer Vision and Pattern Recognition, pages 8748–8757, 2019.

[21] R Kesten, M Usman, J Houston, T Pandya, K Nadhamuni, A Ferreira, M Yuan, B Low, A Jain, P Ondruska, et al. Lyft level 5 av dataset 2019. *urlhttps://level5. lyft. com/dataset*, 2019.

[22] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11621–11631, 2020.

[23] Xinyu Huang, Peng Wang, Xinjing Cheng, Dingfu Zhou, Qichuan Geng, and Ruigang Yang. The apolloscape open dataset for autonomous driving and its application. *IEEE transactions on pattern analysis and machine intelligence*, 42(10):2702–2719, 2019.

[24] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[25] Nicholas Rhinehart, Rowan McAllister, Kris Kitani, and Sergey Levine. Precog: Prediction conditioned on goals in visual multi-agent settings. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2821–2830, 2019.

[26] Kyoungho Ahn, Hesham Rakha, David K Hale, et al. Multi-modal intelligent traffic signal systems (mmitss) impacts assessment. Technical report, United States. Department of Transportation. Intelligent Transportation . . . , 2015.

[27] Debby Bezzina and James Sayer. Safety pilot model deployment: Test conductor team report. *Report No. DOT HS*, 812(171):18, 2014.

[28] Wei Zhan, Liting Sun, Di Wang, Haojie Shi, Aubrey Clausse, Maximilian Naumann, Julius Kummerle, Hendrik Konigshof, Christoph Stiller, Arnaud de La Fortelle, et al. Interaction dataset: An international, adversarial and cooperative motion dataset in interactive driving scenarios with semantic maps. *arXiv preprint arXiv:1910.03088*, 2019.

[29] Julian Bock, Robert Krajewski, Tobias Moers, Steffen Runde, Lennart Vater, and Lutz Eckstein. The ind dataset: A drone dataset of naturalistic road user trajectories at german intersections. *arXiv preprint arXiv:1911.07602*, 2019.

[30] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B Choy, Philip HS Torr, and Manmohan Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 336–345, 2017.

[31] Nicholas Rhinehart, Kris M Kitani, and Paul 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.

[32] Yuexin Ma, Xinge Zhu, Sibo Zhang, Ruigang Yang, Wenping Wang, and Dinesh Manocha. Trafficpredict: Trajectory prediction for heterogeneous traffic-agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6120–6127, 2019.

[33] Rohan Chandra, Tianrui Guan, Srujan Panuganti, Trisha Mittal, Uttaran Bhattacharya, Aniket Bera, and Dinesh Manocha. Forecasting trajectory and behavior of road-agents using spectral clustering in graph-lstms. *IEEE Robotics and Automation Letters*, 5(3):4882–4890, 2020.

[34] Chenxu Luo, Lin Sun, Dariush Dabiri, and Alan Yuille. Probabilistic multimodal trajectory prediction with lane attention for autonomous vehicles. *arXiv preprint arXiv:2007.02574*, 2020.

[35] Jan Strohbeck, Vasileios Belagiannis, Johannes Müller, Marcel Schreiber, Martin Herrmann, Daniel Wolf, and Michael Buchholz. Multiple trajectory prediction with deep temporal and spatial convolutional neural networks.

[36] Xin Huang, Stephen G McGill, Jonathan A DeCastro, Luke Fletcher, John J Leonard, Brian C Williams, and Guy Rosman. Diversitygan: Diversity-aware vehicle motion prediction via latent semantic sampling. *IEEE Robotics and Automation Letters*, 5(4):5089–5096, 2020.

[37] Christopher M Bishop. Mixture density networks. 1994.

[38] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[39] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347, 2017.

[40] Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural autoregressive flows. In *International Conference on Machine Learning*, pages 2078–2087, 2018.

[41] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[42] Boris Ivanovic, Karen Leung, Edward Schmerling, and Marco Pavone. Multimodal deep generative models for trajectory prediction: A conditional variational autoencoder approach. *IEEE Robotics and Automation Letters*, 6(2):295–302, 2020.

[43] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[44] Christopher Z Mooney. *Monte carlo simulation*. Number 116. Sage, 1997.

[45] George R Terrell and David W Scott. Variable kernel density estimation. *The Annals of Statistics*, pages 1236–1265, 1992.

[46] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 62(2):1805, 2000.

[47] Sergio Casas, Wenjie Luo, and Raquel Urtasun. Intentnet: Learning to predict intention from raw sensor data. In *Conference on Robot Learning*, pages 947–956, 2018.

[48] Wenjie Luo, Bin Yang, and Raquel Urtasun. Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 3569–3577, 2018.

[49] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 961–971, 2016.

[50] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2255–2264, 2018.

[51] Nauman Sohani, Geunseob GS Oh, and Xinpeng Wang. A data-driven, falsification-based model of human driver behavior. In *2020 American Control Conference (ACC)*, pages 3892–3899. IEEE, 2020.

[52] Geunseob Oh and Huei Peng. Impact of traffic lights on trajectory forecasting of human-driven vehicles near signalized intersections. *arXiv preprint arXiv:1906.00486*, 2019.

[53] Amir Sadeghian, Vineet Kosaraju, Ali Sadeghian, Noriaki Hirose, Hamid Rezatofighi, and Silvio Savarese. Sophie: An attentive gan for predicting paths compliant to social and physical constraints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1349–1358, 2019.

[54] Charlie Tang and Russ R Salakhutdinov. Multiple futures prediction. In *Advances in Neural Information Processing Systems*, pages 15424–15434, 2019.

[55] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[56] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Advances in neural information processing systems*, pages 3483–3491, 2015.

[57] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[58] Jiachen Li, Hengbo Ma, and Masayoshi Tomizuka. Interaction-aware multi-agent tracking and probabilistic behavior prediction via adversarial learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6658–6664. IEEE, 2019.

[59] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

[60] Henggang Cui, Vladan Radosavljevic, Fang-Chieh Chou, Tsung-Han Lin, Thi Nguyen, Tzu-Kuo Huang, Jeff Schneider, and Nemanja Djuric. Multimodal trajectory predictions for autonomous driving using deep convolutional networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2090–2096. IEEE, 2019.

[61] Alberto Elfes. Occupancy grids: A probabilistic framework for robot perception and navigation. 1991.

[62] Daniel Meyer-Delius, Maximilian Beinhofer, and Wolfram Burgard. Occupancy grid models for robot mapping in changing environments. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, 2012.

[63] ByeoungDo Kim, Chang Mook Kang, Jaekyum Kim, Seung Hi Lee, Chung Choo Chung, and Jun Won Choi. Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 399–404. IEEE, 2017.

[64] Seong Hyeon Park, ByeongDo Kim, Chang Mook Kang, Chung Choo Chung, and Jun Won Choi. Sequence-to-sequence prediction of vehicle trajectory via lstm encoder-decoder architecture. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1672–1678. IEEE, 2018.

[65] Nima Mohajerin and Mohsen Rohani. Multi-step prediction of occupancy grid maps with recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10600–10608, 2019.

[66] Joey Hong, Benjamin Sapp, and James Philbin. Rules of the road: Predicting driving behavior with a convolutional model of semantic interactions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8454–8462, 2019.

[67] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[68] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. *Advances in neural information processing systems*, 28:2980–2988, 2015.

[69] Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. Variational graph recurrent neural networks. In *Advances in neural information processing systems*, pages 10701–10711, 2019.

[70] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[71] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[72] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

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

[74] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[75] Luca Bertinetto, João F Henriques, Jack Valmadre, Philip Torr, and Andrea Vedaldi. Learning feed-forward one-shot learners. *Advances in neural information processing systems*, 29:523–531, 2016.

[76] Joseph Suarez. Language modeling with recurrent highway hypernetworks. *Advances in neural information processing systems*, 30:3267–3276, 2017.

[77] Andrew Brock, Theodore Lim, James Millar Ritchie, and Nicholas J Weston. Smash: One-shot model architecture search through hypernetworks. In *6th International Conference on Learning Representations*, 2018.

[78] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. In *International Conference on Learning Representations*, 2018.

[79] Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F Grewe. Continual learning with hypernetworks. In *International Conference on Learning Representations*, 2019.

[80] Jonathan Lorraine and David Duvenaud. Stochastic hyperparameter optimization through hypernetworks. *arXiv preprint arXiv:1802.09419*, 2018.

[81] Tomer Galanti and Lior Wolf. On the modularity of hypernetworks. *Advances in Neural Information Processing Systems*, 33, 2020.

[82] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[83] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[84] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in neural information processing systems*, pages 10215–10224, 2018.

[85] You Lu and Bert Huang. Structured output learning with conditional generative flows. In *AAAI*, pages 5005–5012, 2020.

[86] Nicola De Cao, Wilker Aziz, and Ivan Titov. Block neural autoregressive flow. In *Uncertainty in Artificial Intelligence*, pages 1263–1273. PMLR, 2020.

[87] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in neural information processing systems*, pages 4743–4751, 2016.

[88] Aaron van den Oord, Oriol Vinyals, et al. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pages 6306–6315, 2017.

[89] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538, 2015.

[90] Rianne van den Berg, Leonard Hasenclever, Jakub M Tomczak, and Max Welling. Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649*, 2018.

[91] Z Ziegler and A Rush. Latent normalizing flows for discrete sequences. In *International Conference of Machine Learning*, 2019.

[92] Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard Hovy. Flowseq: Non-autoregressive conditional sequence generation with generative flow. *arXiv preprint arXiv:1909.02480*, 2019.

[93] Emiel Hoogeboom, Rianne Van Den Berg, and Max Welling. Emerging convolutions for generative normalizing flows. In *International Conference on Machine Learning*, pages 2771–2780, 2019.

[94] Jens Behrmann, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In *International Conference on Machine Learning*, pages 573–582. PMLR, 2019.

[95] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. In *Advances in Neural Information Processing Systems*, pages 9605–9616, 2018.

[96] Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

[97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[98] Ruizhi Deng, Bo Chang, Marcus A Brubaker, Greg Mori, and Andreas Lehrmann. Modeling continuous stochastic processes with dynamic normalizing flows. *arXiv preprint arXiv:2002.10516*, 2020.

[99] Alexander Radovic, Jiawei He, Janahan Ramanan, Marcus A Brubaker, and Andreas Lehrmann. Agent forecasting at flexible horizons using ode flows. In *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*, 2021.

[100] Yuning Chai, Benjamin Sapp, Mayank Bansal, and Dragomir Anguelov. Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction. *arXiv preprint arXiv:1910.05449*, 2019.

[101] Tung Phan-Minh, Elena Corina Grigore, Freddy A Boulton, Oscar Beijbom, and Eric M Wolff. Covernet: Multimodal behavior prediction using trajectory sets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14074–14083, 2020.

[102] Steven G Stradling. Transport user needs and marketing public transport. In *Proceedings of the Institution of Civil Engineers-Municipal Engineer*, volume 151, pages 23–28. Thomas Telford Ltd, 2002.

[103] David Schrank, Bill Eisele, Tim Lomax, and Jim Bak. 2015 urban mobility scorecard. 2015.

[104] Nir Sharaby and Yoram Shiftan. The impact of fare integration on travel behavior and transit ridership. *Transport Policy*, 21:63–70, 2012.

[105] Natasha Merat, Ruth Madigan, and Sina Nordhoff. Human factors, user requirements, and user acceptance of ride-sharing in automated vehicles. 2017.

[106] Brandon Schoettle and Michael Sivak. Public opinion about self-driving vehicles in china, india, japan, the us, the uk, and australia. Technical report, University of Michigan, Ann Arbor, Transportation Research Institute, 2014.

[107] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[108] Donald E Kirk. Optimal control theory: An introduction (prentice-hall networks series). 1970.

[109] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995.

[110] Felicitas Mensing, Rochdi Trigui, and Eric Bideaux. Vehicle trajectory optimization for application in eco-driving. In *2011 IEEE Vehicle Power and Propulsion Conference*, pages 1–6. IEEE, 2011.

[111] Yuguang Wei, Cafer Avcı, Jiangtao Liu, Baloka Belezamo, Nizamettin Aydın, Pengfei Taylor Li, and Xuesong Zhou. Dynamic programming-based multi-vehicle longitudinal trajectory optimization with simplified car following models. *Transportation research part B: methodological*, 106:102–129, 2017.

[112] Geunseob Oh and Huei Peng. Eco-driving at signalized intersections: What is possible in the real-world? In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3674–3679. IEEE, 2018.

[113] Bart Saerens. Optimal control based eco-driving. *Theoretical Approach and Practical Applications. Heverlee: Katholieke Universiteit Leuven*, 2012.

[114] Yiyong Xiao and Abdullah Konak. A genetic algorithm with exact dynamic programming for the green vehicle routing & scheduling problem. *Journal of Cleaner Production*, 167:1450–1463, 2017.

[115] Marlin Wolf Ulmer. *Approximate dynamic programming for dynamic vehicle routing*, volume 61. Springer, 2017.

[116] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.

[117] Frank L Lewis, Draguna Vrabie, and Vassilis L Syrmos. *Optimal control*. John Wiley & Sons, 2012.

[118] Vibhor L Bageshwar, William L Garrard, and Rajesh Rajamani. Model predictive control of transitional maneuvers for adaptive cruise control vehicles. *IEEE Transactions on Vehicular Technology*, 53(5):1573–1585, 2004.

[119] Paolo Falcone, Francesco Borrelli, Jahan Asgari, H Eric Tseng, and Davor Hrovat. A model predictive control approach for combined braking and steering in autonomous vehicles. In *2007 Mediterranean Conference on Control & Automation*, pages 1–6. IEEE, 2007.

[120] JM Park, DW Kim, YS Yoon, HJ Kim, and KS Yi. Obstacle avoidance of autonomous vehicles based on model predictive control. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 223(12):1499–1516, 2009.

[121] Baisravan HomChaudhuri, Ardalan Vahidi, and Pierluigi Pisu. A fuel economic model predictive control strategy for a group of connected vehicles in urban roads. In *2015 American Control Conference (ACC)*, pages 2741–2746. IEEE, 2015.

[122] Md Abdus Samad Kamal, Masakazu Mukai, Junichi Murata, and Taketoshi Kawabe. Model predictive control of vehicles on urban roads for improved fuel economy. *IEEE Transactions on control systems technology*, 21(3):831–841, 2012.

[123] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

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

[125] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in neural information processing systems*, pages 3338–3346, 2014.

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

[127] Maxime Bouton, Akansel Cosgun, and Mykel J Kochenderfer. Belief state planning for autonomously navigating urban intersections. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 825–830. IEEE, 2017.

[128] Huile Xu, Yi Zhang, Li Li, and Weixia Li. Cooperative driving at unsignalized intersections using tree search. *IEEE Transactions on Intelligent Transportation Systems*, 21(11):4563–4571, 2019.

[129] Adrien Couëtoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. Continuous upper confidence trees. In *International Conference on Learning and Intelligent Optimization*, pages 433–445. Springer, 2011.

[130] Jongmin Lee, Wonseok Jeon, Geon-Hyeong Kim, and Kee-Eung Kim. Monte-carlo tree search in continuous action spaces with value gradients. In *AAAI*, pages 4561–4568, 2020.

[131] Jongmin Lee, Geon-Hyeong Kim, Pascal Poupart, and Kee-Eung Kim. Monte-carlo tree search for constrained pomdps. *Advances in Neural Information Processing Systems*, 31:7923–7932, 2018.

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

[133] Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

[134] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[135] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[136] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[137] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[138] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR (Poster)*, 2016.

[139] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870, 2018.

[140] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

[141] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[142] Sham M Kakade. A natural policy gradient. *Advances in neural information processing systems*, 14:1531–1538, 2001.

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

[144] Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, TB Dhruva, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. In *International Conference on Learning Representations*, 2018.

[145] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pages 12519–12530, 2019.

[146] Rowan McAllister and Carl Edward Rasmussen. Data-efficient reinforcement learning in continuous state-action gaussian-pomdps. *Advances in Neural Information Processing Systems*, 30:2040–2049, 2017.

[147] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[148] Gregory Kahn, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Plato: Policy learning using adaptive trajectory optimization. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3342–3349. IEEE, 2017.

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

[150] Zdravko I Botev, Dirk P Kroese, Reuven Y Rubinstein, and Pierre L'Ecuyer. The cross-entropy method for optimization. In *Handbook of statistics*, volume 31, pages 35–59. Elsevier, 2013.

[151] Gordon Frank Newell. A simplified car-following theory: a lower order model. *Transportation Research Part B: Methodological*, 36(3):195–205, 2002.

[152] Peter G Gipps. A behavioural car-following model for computer simulation. *Transportation Research Part B: Methodological*, 15(2):105–111, 1981.

[153] Geunseob Oh, Nauman Sohani, and Huei Peng. Control-theoretic evaluation of policy in sequential decision making via data-driven differential game. 2020.

[154] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. *Advances in Neural Information Processing Systems*, 30:2775–2785, 2017.

[155] Daniel Krajzewicz, Georg Hertkorn, Christian Rössel, and Peter Wagner. Sumo (simulation of urban mobility)-an open-source traffic simulation. In *Proceedings of the 4th middle East Symposium on Simulation and Modelling (MESM20002)*, pages 183–187, 2002.

[156] Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 1–8, 2006.

[157] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[158] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[159] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[160] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[161] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, et al. Conformer: Convolution-augmented transformer for speech recognition. *arXiv preprint arXiv:2005.08100*, 2020.

[162] Cunjun Yu, Xiao Ma, Jiawei Ren, Haiyu Zhao, and Shuai Yi. Spatio-temporal graph transformer networks for pedestrian trajectory prediction. In *European Conference on Computer Vision*, pages 507–523. Springer, 2020.

[163] Xin Li, Xiaowen Ying, and Mooi Choo Chuah. Grip: Graph-based interaction-aware trajectory prediction. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3960–3966. IEEE, 2019.

[164] Rohan Chandra, Tianrui Guan, Srujan Panuganti, Trisha Mittal, Uttaran Bhattacharya, Aniket Bera, and Dinesh Manocha. Forecasting trajectory and behavior of road-agents using spectral clustering in graph-lstms. *IEEE Robotics and Automation Letters*, 5(3):4882–4890, 2020.

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

[166] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. In *IJCAI*, volume 7, pages 2586–2591, 2007.

[167] Jack N Barkenbus. Eco-driving: An overlooked climate change initiative. *Energy policy*, 38(2):762–769, 2010.

[168] Geunseob Oh, David J Leblanc, and Huei Peng. Vehicle energy dataset (ved), a large-scale dataset for vehicle energy consumption research. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[169] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2019.