Synthesis and Evaluation of Automated Vehicles

by

Songan Zhang

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Mechanical Engineering)
in the University of Michigan
2021

Doctoral Committee:

Professor Huei Peng, Chair
Professor Brent Gillespie
Assistant Professor Alex Gorodetsky
Professor Jing Sun

Songan Zhang

songanz@umich.edu

ORCID iD: 0000-0002-3238-5406

Acknowledgements

Four years have passed since 2016, the start of my journey of pursuing the Ph.D. degree. During this journey, I have always been full of passion and curiosity. As I reach the end of this road, I would like to extend my sincere and heartfelt obligation towards all the personages who have helped me in this endeavor. Without their active guidance, help, cooperation, and encouragement, I would not make headway in this journey.

First, I would first like to thank my advisor, Professor Huei Peng, whose expertise was invaluable in formulating the research questions and methodology. His insightful feedback pushed me to sharpen my thinking and brought my work to a higher level. Besides, he also taught me how to balance my life, family, and research and gave me sincere suggestions about my career choice. His noble example inspired me to greater efforts, and I am grateful to have a chance to work with him.

Second, I would like to acknowledge my committee members: Professor Jing Sun, Professor Brent Gillespie, and Professor Alex Gorodetsky. Without your help, I could never reach the point where I stand now. The meetings with you were an amazing source of information and ideas, and I received a great deal of help.

I want to thank my brothers and sisters in the Vehicle Dynamic Lab. Ding, Shaobing, Pingping, Yiqun, Steven, Yuxiao, Ziheng, Xianan, Su-Yang, Geunsoab, Nauman, Bopi, Xingpeng, Yuanxin, Minghan, Lu, Zhong, and Juhui, without you guys, my life in Ann Arbor will never be so colorful.

I would also like to thank my husband, Fucong Wang, for your unconditional love. During this journey, you are always by my side and support me whenever and wherever.

Finally, my greatest appreciation goes to my parents, Weilong Zhang and Ping Hu. Thank you for always being there for me. Without the two of you, I do not know where I would be. If I have learned anything while being away from you, it is that you are the most important people in my life, and I love you both more than anything.

Table of Contents

Acknowledgements ii

List of Tables vi

List of Figures vii

List of Abbreviations ix

Abstract xi

Chapter 1 Introduction 1

1.1 Background and Motivation Introduction 1

1.2 Literature Review 5

1.2.1 Synthesis Approaches for the Decision-making of Automated Vehicles 5

1.2.2 Evaluation Approaches for Automated Vehicles 8

1.3 AV Synthesis Problem 14

1.4 AV Evaluation Problem 15

1.5 Contributions 17

1.6 Outline of the Dissertation 18

Chapter 2 Synthesis of the Autonomous Vehicle's Policy Using Reinforcement Learning 19

2.1 Literature Reviews on Discretionary Lane Change Decision-making Approaches 19

2.2 Reinforcement Learning Fundamentals 20

2.2.1 Dynamic Programming 21

2.2.2 Temporal Difference Method 22

2.2.3 Policy Gradient Methods 25

2.2.4 Exploration-exploitation Trade-off and Current Methods 26

2.3 Model-based Exploration of Reinforcement Learning 27

2.3.1 Simulator and Reward Function 28

2.3.2 Environment model 29

2.3.3 Intrinsic reward 32

2.3.4 Model and policy update strategy 34

2.3.5 Baseline Exploration Method 35

2.3.6 Training Setups 36

2.4 Training Results 36
2.5 Summary 39
Chapter 3 Evaluation of the Autonomous Vehicle's Policy Using Subset Simulation 40
3.1 Literature Reviews on Evaluation Approaches and Their Limitations 40
3.1.1 Importance Sampling Method 41
3.1.2 Limitations and Motivations 43
3.2 Variance Reduction Techniques: Subset Simulation 45
3.2.1 Subset Simulation Algorithm 45
3.2.2 Modified Metropolis Algorithm 47
3.3 Naturalistic Driving Data and Environment Model 50
3.3.1 Naturalistic Driving Data 50
3.3.2 Environment Model 54
3.3.3 Rosenblatt Transformation 58
3.4 Simulation and Results 59
3.4.1 Advanced Emergency Braking System under Test 59
3.4.2 Simulation Setups 60
3.4.3 Evaluation Results 62
3.5 Summary 64
Chapter 4 Evaluation of the Autonomous Vehicle's Policy Using Learning-Based Approach 66
4.1 Motivations 66
4.1.1 Difficulties of implementing IS and SS with different environments 66
4.1.2 Decision-making System Evaluation 67
4.2 Literature Reviews on Attacking Deep Neural Networks 68
4.2.1 Attacks on Deep Neural Network 68
4.2.2 Attacks on Deep Reinforcement Learning 69
4.3 Methodology 70
4.3.1 Markov Game 71
4.3.2 Attacker's Training Environment 72
4.3.3 Socially Acceptable Attacks 72
4.3.4 Reward Considering Socially Acceptable Attack 73
4.4 Training Setups 78
4.4.1 The Victim AV under test 78
4.4.2 Training Setups for the Attacker 79
4.5 Attacker's Training Results and The Victim AV Evaluation Results 80

4.6 Summary 82
Chapter 5 Meta Reinforcement Learning for Synthesis Adaptive Decision-making Policy 83
5.1 Motivation and Objective 83
5.2 Literature Review and Meta Reinforcement Learning Preliminary 84
5.2.1 Mathematical Formulation 84
5.2.2 Literature Review 85
5.3 Efficient Off-policy Meta Reinforcement Learning Method 87
5.3.1 Learning and Modeling Latent Contexts 88
5.3.2 Advanced Exploration via Posterior Sampling 89
5.3.3 Off-Policy Meta-Reinforcement Learning 90
5.4 Discretionary Lane Change Environment Distribution 91
5.4.1 Environments with Attackers 91
5.4.2 IDM-Mobil Driver Model Environments 92
5.5 Training Setup 94
5.5.1 Baselines for the Meta Training and Adaptation 94
5.5.2 Training Hyperparameters 95
5.6 Results 96
5.6.1 Training Results 96
5.6.2 Evaluation Results 97
5.7 Summary 100
Chapter 6 Conclusions and Future Works 101
6.1 Conclusions 101
6.2 Future Works 103
6.2.1 Designing Robust Decision-making Systems of Other AV Scenarios 103
6.2.2 Online Monitoring Environment Changes 103
6.2.3 Extrapolate rather than Interpolate 104
Bibliography 105

List of Tables

Table 1.1 Summary of Levels of Automation for On-road Vehicles....2
Table 1.2 Automotive Companies Announced AV Plans and Status at 2020....3
Table 1.3 Serious Accidents involving AVs [11]....4
Table 1.4 Major Naturalistic Field Operational Test Databases....10
Table 2.1 Implementation Hyperparameters....36
Table 3.1 Lane Change Model Features....52
Table 3.2 Correlation Matrix of the 8 Variables....53
Table 3.3 Parameters for the Subset Simulation....62
Table 3.4 Evaluation Results Summary....64
Table 4.1 Reward Design for Responsibility-sensitive Attack....77
Table 4.2 Hyperparameters for Training the Attacker....79
Table 4.3 Number of Crashes during Evaluation....81
Table 5.1 Mobil parameters for different driver behaviors....94
Table 5.2 Implementation Hyperparameters for PEARL....95
Table 5.3 Crash Rate with Different Numbers of Data in the Attacker Environments....99
Table 5.4 Crash Rate with Different Numbers of Data in the IDM-Mobil Environments....99

List of Figures

Figure 1.1 Autonomous Vehicle Technology Companies [10] 3
Figure 1.2 Iterative Vehicle Dynamic Control Evaluation Process [42] 9
Figure 1.3 The Procedure of the Accelerated Evaluation [69] 12
Figure 1.4 Overview of the Typical Hierarchical Architecture of AVs [12] 16
Figure 2.1 Comparison of the backup diagrams of Monte-Carlo, Temporal-Difference learning, and Dynamic Programming for state value functions [83] 24
Figure 2.2 Different Types of Objective Functions for the Policy Gradient Methods [86] 25
Figure 2.3 Model-based Exploration 28
Figure 2.4 Three Lane Highway Simulator. 28
Figure 2.5 Conditional variational auto-encoder model with the vehicle kinematics model 31
Figure 2.6 Intrinsic Reward 34
Figure 2.7 Training procedure for CVAE and DDQN 35
Figure 2.8 Average reward from evaluation roll-outs confidence bound 37
Figure 2.9 Average reward of last 50 episodes with the confidence bound 37
Figure 2.10 Average intrinsic reward and the CVAE training loss over training iterations 37
Figure 2.11 Animation result of ε-greedy method 39
Figure 2.12 Animation result of model-based exploration method 39
Figure 3.1 Subset Simulation Process 47
Figure 3.2 Schematic Diagram of Modified Metropolis Algorithm 48
Figure 3.3 Lane Change Scenario Features 50
Figure 3.4 Data Processing for Querying the Lane Change Data 52
Figure 3.5 2nd Order Polynomial Fitting Examples 52
Figure 3.6 Schematic Diagram for the GMM, GGMM, and BGGMM 55
Figure 3.7 AIC and BIC Value of the BGGMM with Component Number from 3 to 20 56
Figure 3.8 BGGMM Fitting Results (Marginal Distribution PDF) for 8 Variables 58
Figure 3.9 Layout of the AV Control System [5] 59

Figure 3.10 Samples Expended by SS using the Baseline 3-variable Model.....63
Figure 3.11 Samples Expended by SS using the 8-variable BGGMM Model.....64
Figure 4.1 Sequential Search of Danger Regions 67
Figure 4.2 Generating Adversarial Patches against YOLOv2 [138] 69
Figure 4.3 Attacks on Deep Reinforcement Learning 70
Figure 4.4 Attacker's Training Environment 72
Figure 4.5 No Car on the Lane Marker 75
Figure 4.6 Only One Car on the Lane Marker and Crash with the Car in the Target Lane 75
Figure 4.7 Both Cars are on the Different Lane Markers.....76
Figure 4.8 Three lane highway simulator. The blue box: the AV; red boxes: 6 nearest surrounding vehicles; empty boxes: unobserved surrounding vehicles 78
Figure 4.9 Training Curve of the Attacker 80
Figure 4.10 An example of the AV-responsible crash. 81
Figure 5.1 Illustrating the Inner and Outer Loops of Training [159].....85
Figure 5.2 Multi-MDPs adaptation problem as a POMDP problem 87
Figure 5.3 Collected experience can then be used to update the belief during adaptation [169] ..89
Figure 5.4 Examples of environments with different numbers of attackers (red boxes) 92
Figure 5.5 The highway-env environment [171] 92
Figure 5.6 Meta Testing Average Returns during Meta Training in Attacker Environments.....96
Figure 5.7 Meta Testing Average Returns during Meta Training in IDM-Mobil Environments..97
Figure 5.8 Evaluation Average Returns 98
Figure 6.1 Procedure for developing an AV's decision-making system 102

List of Abbreviations

A2C	Advantage Actor Critic
AC	Actor-Critic
ACC	Adaptive Cruise Control
AEB	Autonomous Emergency Braking
AIC	Akaike Information Criterion
AV	Automated Vehicle
BEE	Best Evasive Effort
BGGMM	Bounded Generalized Gaussian Mixture Model
BIC	Bayesian Information Criterion
c.o.v.	Coefficient of Variation
CDF	Cumulative Distribution Function
CE	Cross Entropy
CMC	Crude Monte Carlo
C-NCAP	China New Car Assessment Program
CNN	Convolutional Neural Network
CV	Connected Vehicle
CVAE	Conditional Variational Auto-Encoder
DDQN	Double Deep Q-Network
DNN	Deep Neural Network
DP	Dynamic Programming
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
EDR	Event Data Recorder
ELBO	Evidence Lower BOund
FC	Failure Code
FMVSS	Federal Motor Vehicle Safety Standards
FSM	Finite State Machines
GAIL	Generative Adversarial Imitation Learning
GES	General Estimates System
GGMM	Generalized Gaussian Mixture Model
GMM	Gaussian Mixture Model
HV	Human-controlled Vehicle
IDM	Intelligent Driver Model
IS	Importance Sampling

ISD	Importance Sampling Distribution
IVBSS	Integrated Vehicle-Based Safety Systems
KB	Knowledge Base
MAML	Model Agnostic Meta-Learning
MC	Monte Carlo
MCMC	Markov Chain Monte Carlo
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
MEE	Minimal Evasive Effort
MLD	Mixed Logical Dynamical
MMA	Modified Metropolis Algorithm
MOT	Moving Objects Tracking
MPC	Model Predictive Control
MRL	Meta Reinforcement Learning
NCAP	New Car Assessment Program
N-FOT	Naturalistic Field Operational Test
NHTSA	National Highway Traffic Safety Administration
NMVCCS	National Motor Vehicle Crash Causation Survey
PCPO	Parallel Constrained Policy Optimization
PDF	Probability Density Function
PEARL	Probabilistic Embeddings for Actor-critic RL
PG	Policy Gradient
POMDP	Partially Observed Markov Decision Process
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RSS	Responsibility-Sensitive Safety
SAA	Socially Acceptable Attack
SAC	Soft Actor-Critic
SAE	Society of Automobile Engineers
SAS	Safety Assist Score
SIS	Sequential Importance Sampling
SMM	Student's-t mixture model
SPMD	Safety Pilot Model Deployment
SS	Subset Simulation
TD	Temporal Difference
TRPO	Trust Region Policy Optimization
TSD	Traffic Signalization Detection
TTC	Time-To-Collision
V2V	Vehicle-to-Vehicle
VAE	Variational Auto-Encoder

Abstract

This dissertation focuses on the synthesis of a decision-making system for Automated Vehicles (AVs), and then evaluates the safety and robustness of the system with an eye toward improving the system design.

We begin with a synthesis of an AV's decision-making system in a specific driving environment. We model the environment as a Markov Decision Process (MDP), with the goal of determining the optimal strategy (that is, policy) for this particular MDP. We propose a novel Reinforcement Learning (RL) method using model-based exploration. This method allows the training agent to explore the MDP state space by maximizing the notion of an agent's surprise about its experiences via intrinsic motivation. The optimal strategy will be deemed to be a global-optimal policy by which the AV can travel more efficiently.

We then evaluate the decision-making system in a naturalistic driving environment. We focus on lane change maneuvers, modeling the differences between AVs and Human-controlled Vehicles (HVs) using the Safety Pilot Model Deployment Program's naturalistic driving data. The probability of crashes serves as the primary metric for evaluating the safety of AV systems. In general, testing a system in a naturalistic driving environment is time-consuming and not cost-effective. To overcome this problem, we propose an accelerated evaluation method called Subset Simulation (SS), which can significantly reduce evaluation time and beat the baseline Importance Sampling (IS) method. This technique is not only capable of evaluating a system with a high-dimension state space, but also has the potential to conduct evaluations of more complicated systems (e.g., object detection systems).

The SS method is limited, however, in that the "danger regions" are searched only as the test procedure unfolds. If the environmental statistics change, the crash rate cannot be estimated accurately. Therefore, we prefer to evaluate the decision-making system without including the environmental statistics. To this end, we propose an evaluation method based on the two-player Markov game. We introduce an attacker into the environment which keeps "attacking" the AV in a socially acceptable fashion. The attacker tries to lure the AV into AV-responsible crashes (as

opposed to “crazy” crashes). Once the attacker has completed training, the AV is evaluated by introducing the attacker. The crash rate of the system then becomes 50 times greater in the environment with the attacker, which allows the system to register fatal flaws in the original training environment design.

Introducing attackers capable of generating socially acceptable attacks makes the behavior of the surrounding vehicles more diverse. Our goal is to improve the original policy so as to design a safe and robust decision-making system under situations with different types of drivers in the environment, different traffic densities, and differing numbers of total surrounding vehicles. We tackle this problem by implementing the state-of-the-art Meta-Reinforcement Learning (MRL) method to train an agent to quickly adapt to different environments with limited data. The MRL-trained policy can significantly decrease the crash rate with a small amount of data across different environments. This technique has tremendous potential for helping the AV quickly adapt to varying conditions such as different locations, weather, and lighting.

Chapter 1 Introduction

1.1 Background and Motivation Introduction

Automated Vehicle (AV) is a very active research area during the past decade. From 2004 to 2013, the U.S. Department of Defense's research arm, DARPA, sponsored the "Grand Challenge" and later the "Urban Challenge," which played a key role in creating excitement and accelerating the development of AV technologies. In 2014, as shown in Table 1.1, the Society of Automobile Engineers (SAE) defined six levels of automated driving in their document J3016 [1]. The key features of an AV are the ability to monitor the driving environment, control steering, brake and throttle, the capability to handle a variety of driving situations, and the need for a human driver as the fallback. This document was issued, in part, to speed up the delivery of an initial regulatory framework and a practice to guide the automotive companies in the safe design, development, testing, and deployment of AVs. Perhaps more importantly, this 6-level definition shows an evolutionary rather than a revolutionary roadmap for technology deployment.

Several government documents were also published around 2014, including the U.S. National Highway Traffic Safety Administration level (NHTSA level) [2] and the Germany Federal Highway Research Institute level (BASt level) [3], which are also listed in Table 1.1. As further explained in [4], level 2 automation is known as a "hands off" feature where the automated system takes full control of the vehicle, but the driver must monitor the driving and serve as the fallback. Level 3 automation is known as "eyes off" where the driver can safely turn their attention away from the driving tasks. At level 4 automation, no driver attention is required. Therefore it is a "mind off" system. And at level 5 automation, no human intervention is required for all scenarios. Some researchers believe Level 5 is just an aspirational goal and cannot be achieved.

Table 1.1 Summary of Levels of Automation for On-road Vehicles

SAE level	SAE name	Execution of Steering and acceleration/ deceleration	Monitoring of driving environment	Fallback performance of dynamic driving task	System capability (driving modes)	BASt level	NHTSA level
0	No Automation	Human	Human	Human	n/a	Driver only	0
1	Driver Assistance	Human and system	Human	Human	Some driving modes	Assisted	1
2	Partial Automation	System	Human	Human	Some driving modes	Partially automated	2
3	Conditional Automation	System	System	Human	Some driving modes	Highly automated	3
4	High Automation	System	System	System	Some driving modes	Fully automated	3/4
5	Full Automation	System	System	System	All driving modes	n/a	

Automotive companies widely follow the SAE's definition, and most took an evolutionary roadmap, while many "tech companies" seem to embrace the revolutionary approach. Zhao et al. [5] summarized the AV production plans of several major car companies back in 2016. Four years later, few had delivered according to their plans. Table 1.2 lists the original AV production plans and the status of several major car manufacturers. Most of the AVs on the markets today are lower level (Levels 1-2) automated vehicles with Adaptive Cruise Control (ACC) and sometimes lane-keeping assist. Currently, according to [4], the only vehicle that is generally accepted to be at level 3 automation is the Audi A8 (equipped with the Traffic Jam Pilot), which only works on congested highways. Tesla has blurred the lines with recent software updates [6]. Most researchers agree that the Tesla "Autopilot" system (which has been denounced by the German and South Korean authorities as confusing and misleading to the general public) is only at Level-2. Still, the marketing language promised that "full self-driving" is coming in future over-the-air updates, which seems to imply they will have Level-5 capabilities. Technology companies like Google, Amazon, Apple, Baidu, and Intel also have high-level AV programs. Starting in the mid-2010s, start-ups joined the race. Here, we showed some of these companies in Figure 1.1.

Table 1.2 Automotive Companies Announced AV Plans and Status at 2020

Manufacturer	SAE level	Original plan of launch year	First Model	Current status in 2020	
				Status	Launch year
Volvo	2	2015	XC90	Level 2: Pilot Assist	2018
Audi	2-3	2016	A8	Level 2-3[1] [7]: Traffic Jam Pilot	2018
BMW	2-3	2017	5 Series	Level 2: Active Driving Assistant	2017
Mercedes Benz	2-3	2017	E-class S-class	Level 2: Drive Pilot	2017
Volkswagen	2	2017	Passat	Level 2: Travel Assist	2017
Ford	2-3	2017	Fusion	Level 2: Co-Pilot 360	2019
General Motors	2-3	2018	XTS	Level 2: SuperCruise	2018
Nissan	3	2020	Leaf	Level 2-3[2] [8][9]: ProPilot, Japan market only	2020
Lexus	3	2020	LS	Planned for service during the 2020 Olympic Games	-

Figure 1.1 Autonomous Vehicle Technology Companies [10]

[1] Audi equipped the A8 with all the components necessary to make Traffic Jam Pilot work, but it had not enabled the feature due to the current regulation, claimed by Audi.

[2] Different news editors from Forbes have different opinions on this level. And after reviewing both articles, we think the automation level of Nissan leaf is between level 2 to level 3.

As can be seen from Table 1.2, most of the AVs on the market today are at SAE level 2. The difficulties for AVs to "level up" to SAE Level 3 or above come from two perspectives. First, there must be a trust-worthy **evaluation** method for these AV systems. Type approval is required in some markets like in EU or China, and desired in the US to address the questions and concerns of public trust. We list several fatal accidents, under the control of an AV system, in Table 1.3. One of the goals of this research is to develop evaluation methods for the AV system before their wide deployments on the public road so that they are safer.

Table 1.3 Serious Accidents involving AVs [11]

Date	System manufacturer	Vehicle type	SAE level	Location	Fatality
20 Jan 2016	Tesla (Autopilot)	Model S	2	Hebei, China	1: Driver
7 May 2016	Tesla (Autopilot)	Model S	2	Florida, U.S.	1: Driver
18 Mar 2018	Uber	Upfitted Volvo	3	Arizona, U.S.	1: Pedestrian
23 Mar 2018	Tesla (Autopilot)	Model X	2	California, U.S.	1: Driver
1 Mar 2019	Tesla (Autopilot)	Model 3	2	Florida, U.S.	1: Driver
19 Sep 2019	Tesla (Autopilot)	Model 3	2	Florida, U.S.	1: Driver

Another difficulty lies in the **synthesis** of high-level AV systems. The main difference between SAE level 2 and level 3 automation is that whether the system can monitor the driving environment [1] reliably enough so that a human fallback is not needed. The presence of a driver monitoring system is not enough to trigger an uptick in the automation level. At level 3, the system needs to do what a driver does: keep an eye on all factors that might affect safety and deal with them. The driver on a level 3 AV does not need to pay attention to the road continuously. Level 3 AVs need a perception system and a reliable decision-making system, which can fulfill path planning and motion planning functions [12].

This dissertation will focus on the efficient synthesis of the decision-making system of an AV. We will present new ideas and results that contribute to both the evaluation and synthesis of AVs at level 2 or higher.

1.2 Literature Review

The field of synthesis and evaluation of the Automated Vehicle (AV) system has a rich history, with early demonstrations in the 1990s [12] and continuous improvement. In this section, past approaches for both synthesis and evaluation problems are reviewed.

1.2.1 Synthesis Approaches for the Decision-making of Automated Vehicles

The autonomy system of AVs is typically organized into at least two parts: perception and decision-making [12]. The decision-making system usually contains several sub-systems: *route planning*, *path planning*, *behavior selection*, *motion planning*, and *control* [12]. In this dissertation, we focus on the behavior selection subsystem of the decision-making system.

The behavior selection subsystem is responsible for choosing the behavior, such as lane selection, intersection handling, traffic light handling, etc. During the DARPA Urban Challenge era, the Finite State Machines (FSM), a **rule-based decision-making method**, was implemented [13] by the Stanford team (which won second place). However, that competition only includes a limited set of urban scenarios, much simpler than what an AV could experience in the real world. For a more complex lane change scenario on highways involving multiple lanes, Kesting et al. [14] developed a general lane-changing model (named as MOBIL) based on the Intelligent Driver Model (IDM) [15]. This MOBIL model analyzes the potential acceleration of the ego car and surrounding cars to make decisions of whether to change lanes or not. This method depends on the IDM model, and therefore, if the surrounding vehicle's behavior is different from the embedded IDM model, the lane change decision may not work well.

Another type of approach used in designing behavior selection is the **ontology-based method**. Ontology-based techniques studied the frameworks of knowledge representation that can be used to model its concepts and their relations. In [16], Zhao et al. used ontology-based Knowledge Base (KB) to model traffic regulations and sensor data, and the KB is constructed manually. This method requires an accurate world model, including road topologies and traffic rules. Recently, Zhao et al. [17] improved their previous work to use only a small part of the original KB and reduced the computation time. However, a key challenge remains: the KB is still being constructed manually.

The behavior selection problem can also be solved using **optimization-based methods**. Nilsson et al. [18] formulated the decision-making problem of choosing the desired lane and velocity as a Model Predictive Control (MPC) problem. The dynamics are modeled as a Mixed

Logical Dynamical (MLD) system and are solved through MPC using a mixed-integer program. However, the equation to be optimized and cost functions are also designed by hand. Moreover, mixed-integer programming suffers from combinatorial complexity, and the required computational time is strongly affected by the dimension of the problem's state space.

Recently, as more and more data is collected from human driving in the real world, researchers start to utilize the **imitation learning** method to train a model that maps perceptual inputs to control commands. In [19], researchers from NVIDIA trained a Convolutional Neural Network (CNN) to directly map raw images from a front-facing camera to steering angle outputs. With training data from human demonstrations, their system learns to do car following on local roads and highways. In [20], Codevilla et al. proposed command-conditional imitation learning: during training, the commands resolve ambiguities in the perceptuomotor mapping, thus facilitating learning; during testing, the commands serve as a communication channel that can be used to direct the controller. A key obstacle is that imitation learning requires big data. Moreover, the policy learned from human drivers' demonstration can be at most as good as the human driver, while AVs promise to surpass human drivers. Finally, the policy learned from a database may perform poorly in unseen scenarios. Therefore, methods that do not rely on experts' demonstration for supervised learning are desired.

The **Markov Decision Processes (MDP)** is a framework that models the decision making in situations where the outcomes are also affected by exogenous inputs. The MDP can be solved via Dynamic Programming (DP) and Reinforcement Learning (RL). As for the **Dynamic Programming (DP) approach**, Guan et al. [21] deduced the optimal policy using the value iteration method. The learned policy achieves safe and efficient driving. However, using the DP method to solve an MDP optimization problem requires an explicit environment transition model, which is not always available considering the complexity of human driver behaviors. Moreover, the DP method is time-consuming and cannot be implemented in real-time for systems with more than 3-4 states plus control variables with today's computation technology.

The MDP can also be solved using **Reinforcement Learning (RL)**, which does not require an exact mathematical model of the environment transition probability. Reinforcement learning does not require big data collected before training, and it focuses on collecting data from the environment and finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge). RL methods have been successfully implemented in AV

applications. Cao et al. [22] used a Monte Carlo Tree Search (MCTS) method to design a highway exiting planner for autonomous vehicles. Compared to a basic rule-based exiting planner, the RL-training algorithm improved the successful exiting rate by around 50%. Mukadam et al. [23] implemented a Deep Q Network with the Q-masking method to make lane change decisions. By using Q-masking, they were able to incorporate prior knowledge and thus an interface between the higher-level planner and the lower level controller. Nageshrao et al. [24] use the Double Deep Q Network method to design a discretionary lane change planner for a three-lane highway with manually designed exploration strategies and rule-based short-horizon safety checks. Wen et al. [25] used the Parallel Constrained Policy Optimization (PCPO) reinforcement learning method for decision-making of multi-vehicles at an intersection, achieving a safe crossing planner.

To train an optimal decision-making policy for AVs, three essential components are needed. The first is a good simulator or a training environment [26]. If the training environment cannot represent the real environment accurately, the trained policy likely will not work well in real-world driving. The second component is a good reward function. The RL method requires a reward function that can represent the objective of the task well. However, the reward function may not be readily available for realistic applications. Therefore, researchers developed an inverse reinforcement learning method to find the reward function from the expert demonstrations that could represent the expert behavior [27]. Different RL methods will be discussed in more detail in Chapter 2.

In recent years, researchers started to design **end-to-end** systems that optimize all processing steps simultaneously. They argued that end-to-end systems would lead to a smaller system since the perception, decision-making, and motion planning functions are combined. The smaller system can be trained more efficiently. In [28], Kuutti et al. use the CarMaker simulator to train an Advantage Actor-Critic (A2C) network for longitudinal driving. The policy takes sensor readings from the CarMaker simulator as input and the pedal action as the output. Jaritz et al. [29] use the TORCS car racing game as the simulator and train a racing policy that can win the game. The policy takes the camera image from the game as an input and the pedal and steering as the outputs. In this dissertation, we will not consider this end-to-end reinforcement learning method since it requires a perception-built-in simulator.

1.2.2 Evaluation Approaches for Automated Vehicles

The evaluation for vehicle's passive safety (crashworthiness) has been studied for decades, e.g., the Federal Motor Vehicle Safety Standards (FMVSS) [30] from the U.S., the New Car Assessment Program (NCAP) [31] and the United Nations Economic Commission for Europe (UNECE) regulation [32] and China New Car Assessment Program (C-NCAP) [33]. The methods behind those regulation tests are a combination of worst-case scenarios and test matrix evaluation. The **worst-case scenario evaluation method** was developed to test the most challenging cases for a vehicle. In [34] and [35], the authors applied the worst-case evaluation method on rollover and jackknifing cases to evaluate the safety of the vehicle based on a dynamic game theory. In [36], Kou implemented the method to evaluate an integrated chassis control system. In general, the vehicle can be modeled mathematically, and the worst-case evaluation can be considered as an optimization problem to solve for a sequence of control inputs (e.g., a sequence of steering, braking, or pedal inputs) that maximizes a cost function [36].

For **test matrix evaluation methods**, a series of scenarios are first defined. The vehicle then goes through these selected scenarios, one by one. Back in 1973, Moore et al. [37] from General Motor started to use a test matrix evaluation method for vehicle emission systems. To test the safety of vehicles, the choice of scenarios is mainly based on crash databases. A series of research on pre-crash scenarios was conducted in [38]–[40]. As described in [5] and [41], the "44-crashes typology" was developed by General Motors based on the General Estimates System (GES) and National Motor Vehicle Crash Causation Survey (NMVCCS) databases [38]. The authors [41] further used the GES, NMVCCS, and Event Data Recorder (EDR) databases to generate the top five scenario groups: car-following, lane-change, left-turn, intersection crossing, and driving in the opposite direction scenarios.

The text matrix evaluation can be **combined** with the worst-case evaluation method [42], as shown in Figure 1.2. Ungoren et al. identified a worst-case scenario with steering input that looks like a lane change maneuver, which results in rollover for the vehicle with different vehicle dynamic control setups (matrix). As mentioned at the beginning of Section 1.2.2 , the passive safety testing regulation is also designed based on worst-case and test matrix ideas. In Euro NCAP [43], the vehicle needs to go through the full-width rigid barrier crash test, offset deformable barrier crash test, side mobile barrier crash test, side pole crash test, Autonomous Emergency Braking (AEB) test, and the whiplash test to get the overall star rating for adult

occupant protection. The Euro NCAP [43] also designed the Safety Assist Score (SAS) for additional award points. The SAS is based on the electronic stability control test, seatbelt reminder test, speed assistance test, AEB interurban test, and the lane support test.

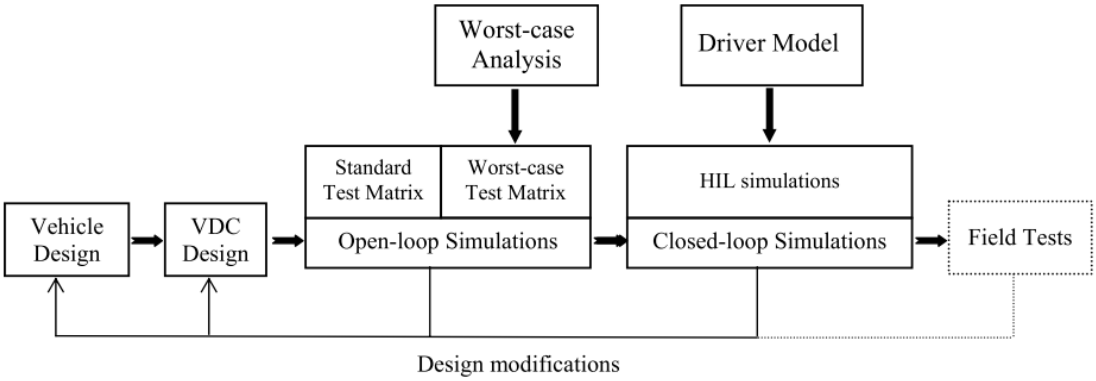


Figure 1.2 Iterative Vehicle Dynamic Control Evaluation Process [42]

Even though the **worst-case evaluation** and **test matrix evaluation** methods have been successfully implemented in traditional vehicle safety tests, there are many difficulties to overcome in testing the safety of AV. For worst-case evaluation methods, they did not consider the probability of occurrence of these worst-case scenarios. Moreover, for different AVs, the worst-case scenarios could be different. Therefore, the results do not offer enough information about the crash rate and risk level in real-world driving and could not provide a fair way to evaluate different vehicles from different companies. For test matrix evaluation methods, the test scenarios are predefined and fixed. Therefore, the control systems may be tweaked to achieve good scores in these tests, but the control systems' performances under broader conditions are not sufficiently evaluated. Moreover, the selected scenarios are usually based on a crash database, in which all the crashes were caused by human drivers. This may not accurately reflect the safety-critical cases for AVs [5].

The AV control systems can also be tested (or verified) by **formal methods**, which are mathematical approaches, to prove or disprove that the system satisfies its requirements (correctness) if it is defined by formal requirements. Formal methods can exhaustively consider uncertainties from initial states, disturbances, and sensor noise. Formal methods contain several different approaches, including *reachability analysis*, *temporal logic model checking*, and

simulation-based falsification. Althoff et al. use *reachability analysis* to verify the safety of AVs [44]–[46]. In [47], they use the *reachability analysis* method to calculate a drivable area under a certain obstacle setup, and the drivable area can be seen as a measure of critical scenarios. Computing the reachability set requires both the environment dynamic model and the AV's dynamic model, which are not always available. The *temporal logic model checking* relies on an exhaustive search of the state space of a finite state system [48]. Therefore, it suffers from the state explosion problem and hard to be implemented in industrial-size systems [49]. In [50], Tuncali used the *simulation-based falsification* approach to search for the initial condition that can falsify the AV system. However, the *simulation-based falsification* approach is a semi-formal method, which checks the formal requirements for each simulation but does not guarantee that the system fulfills the requirement at all states. Therefore, it cannot prove the correctness of the system.

Table 1.4 Major Naturalistic Field Operational Test Databases

Database name	Released by	Labeling	Labeled frames	Sensor
Waymo [51]	Waymo	4 classes 2D/3D	1.2M	5 cameras, 5 lidars
Lyft-Level-5 [52]	Lyft	23 classes 3D	55k	7 cameras (Stereos), GPS/IMU, 3 lidars
nuScenes [53]	Aptiv	23 classes 3D	1.4M	6 camera (Stereos), GPS/IMU, 1 lidar, 1 radar
H3D [54]	Honda	8 classes 3D	1.1M	3 cameras, GPS/IMU, 1 lidar
KITTI [55]	KIT	8 classes 2D/3D	16k	1 lidar, 4 cameras, GPS/IMU

One widely used method for "evaluating" AVs is the **Naturalistic Field Operational Tests** (N-FOTs) [56]. In an N-FOT, the testing vehicles need to be driven in naturalistic conditions over a long time [57]. The most famous N-FOTs project in the U.S. is the Google Waymo AV testing project [58]. Based on the Waymo's Safety Report [58] published in 2018, Waymo's AVs were tested on public roads in 25 cities from 6 states for over 5 million miles (by Jan 2020, the accumulated mileage is 20 million miles [59]). This project allows Waymo engineers to validate the technologies they have developed. By Jan 2018, the company's AVs have encountered 36 crashes. Most of these crashes involved been rear-ended or side-swiped by a human-driven vehicle. Other major N-FOTs projects in the U.S. include the Safety Pilot Model Deployment (SPMD) program [60]–[62] and the Integrated Vehicle-Based Safety Systems

(IVBSS) program [63]. Recent N-FOT databases deliver raw camera and lidar data, many of them with human-generated labels. These data are useful for training object detection and tracking algorithms. Some major N-FOT databases with the raw camera and lidar data are listed in Table 1.4.

The downsides of using N-FOTs for evaluation are obvious. Conducting such N-FOT projects to evaluate the safety of an AV is both time-consuming and expensive. Under naturalistic driving conditions, the probability of encountering conflict scenarios is very low. In 2013, [64] estimated that an N-FOT project could not be conducted with less than 10 million USD. From a newsletter [65] in 2016, Google claimed to pay Arizona drivers 20 USD per hour to test self-driving cars. It was estimated that Google might have spent 2-3 million on test drivers alone. The hardware of each test vehicle costs at least one million dollars. Kalra et al. [66] approximate that one has to test AV for 440 million km (273.4 million miles) to demonstrate that they are safer than human drivers with a 95% confidence level, and this is approximately equivalent to 12.5 years of test driving with 100 AVs. For evaluation proposes, during the N-FOTs period, the control system design of tested AV should not change. Therefore, a more effective evaluation method for testing AV is necessary.

Researchers also built stochastic models based on naturalistic driving data for **Monte Carlo (MC) simulations** to assess AVs' safety. In [67], Yang et al. evaluated collision avoidance systems by building an "errorable" driver model. Jurecki et al. [68] tested drivers' behavior in simulated traffic. Driver reaction time was found to be a function of Time-To-Collision (TTC), which characterizes accident risk situations.

A major benefit of MC simulation is that the simulated cases can be based on naturalistic driving statistics. Moreover, since the tests are conducted through simulations, the cost is much lower than N-FOTs[3]. However, if the MC simulation approach is used directly, the probability of encountering conflict scenarios is still very low.

This is when the **accelerated evaluation** approach is developed. The accelerated evaluation concept was first introduced for AV evaluations by Zhao [5] in 2016. By skewing the statistics of the driver behavior of the surrounding vehicles, the evaluation can focus on higher-

[3] Even though the MC simulator is built based on the naturalistic database, after designing the simulator, different AVs are tested in this simulator and no new data collection is needed.

risk cases and thus saves time, i.e., the evaluation procedure is accelerated. The procedure of accelerated evaluation is shown in Figure 1.3, which includes six steps:

1. Collect a large quantity of naturalistic driving data.
2. Extract scenarios that have potential conflicts between an AV and surrounding human-controlled vehicles.
3. Model the behavior of the surrounding vehicles as a probability distribution $f(\boldsymbol{x})$, where $\boldsymbol{x}$ represents the random variables vector, which captures features of each scenario.
4. Skew this distribution[4] to a $f^*(\boldsymbol{x})$ to emphasize higher-risk situations.
5. Conduct MC tests with the skewed (accelerated) distribution and get the test results.
6. "Skew back" the results to reconstruct the performance of the AV under naturalistic driving conditions.

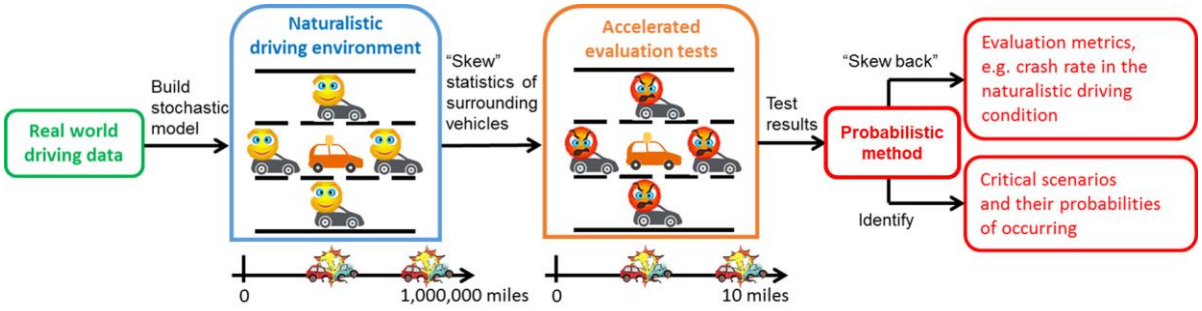


Figure 1.3 The Procedure of the Accelerated Evaluation [69]

The accelerated evaluation approach has been successfully implemented in a lot of studies. The math behind this approach is the **Importance Sampling (IS) method**. In [69], the authors extracted car-following scenarios from the SPMD database and applied the importance sampling method to evaluate the crash, injury, and conflict rates for a simulated AV. In [70], they extended the method to lane change scenarios. They use the cross-entropy approach to search for the optimal accelerated distribution's parameters. Then the probabilities of conflicts, crashes, and injuries are estimated for a tested AV using that distribution, and the achieved

[4] In the original work by Ding et al. [5], they use importance sampling method to calculate an accelerated distribution $f^*(\boldsymbol{x})$ and replace the original $f(\boldsymbol{x})$.

accelerated rate is around 2000 to 20,000. Huang et al. [71] further improved the accelerated evaluation by using piecewise mixture distribution models instead of single parametric distribution models. Simulation results showed that the piecewise mixture distribution outperformed single parametric distribution methods in accuracy and efficiency. O'Kelly et al. [72] implemented the adaptive importance-sampling methods to evaluate systems that employ deep-learning perception and control algorithms and developed a scalable end-to-end autonomous vehicle testing framework.

Although the accelerated evaluation approach is efficient and has a lot of potential in the field of AV testing, importance-sampling is not necessarily the best approach for "skewing" the distribution. First, the IS method suffers from the curse of dimensionality problem. When the environment model has a high dimension, extra care must be taken in the choice of model structure of the IS Distribution (ISD). Otherwise, the estimation may "degenerate," giving results that do not reflect the true results [73]. This is a serious issue for the IS method in the AV evaluation. If the environment model has a small number of random variables, it can be analyzed directly and completely by formal methods. For problems with many random variables, the IS method cannot provide reliable results. This will be expounded in Chapter 3. Another issue of the IS method is that for evaluating the safety of AVs, a common choice for ISD is to "skew" toward the "danger region." However, the "danger region" of a specific AV is not known beforehand. Not knowing the "danger region" means that some searching algorithms need to be used, and a common choice is the cross-entropy method [74]. However, a searching algorithm requires tests for identifying the "danger region," which will drain the accelerating rate. Therefore, a more versatile and advanced method is needed.

Recently, Feng et al. [75], [76] use a **reinforcement learning**-based approach to improve the case searching process in high-dimensional test scenarios. Feng et al. first formulated the test scenario library generation problem as a Markov Decision Process (MDP) in the car-following case. The value function of this MDP is then defined directly by the probability of failure and then trained with Temporal-Difference (TD) reinforcement learning (RL). Although the RL algorithm can help to search the "danger region" on a high-dimensional state space, the authors only implement this method in the car-following scenario. The reason is that in the car-following scenario, avoiding a crash is the rear car's responsibility. In other scenarios, without certainty on the responsibility of the crash, it is not straightforward to find the "danger region." Therefore,

implementing a searching algorithm without naturalistic driving data requires clarifying the responsibility of crashes and conflicts. This will be discussed in detail in Chapter 4.

1.3 AV Synthesis Problem

The objective of our work on the synthesis problem is to build an AV decision-making subsystem that can drive the vehicle safely and efficiently. As explained in Section 1.2.1 , the Reinforcement Learning (RL) method is a promising approach to find the optimal policy in a complicated environment. Therefore, we will focus on using the RL method for synthesis.

To design an optimal decision-making policy, a simulator for the driving environment is needed. First of all, given a complicated environment, solving for the optimal policy is not easy. Therefore, in the first setup of assumption, we assume having a perfect simulator, and we are trying to find the optimal policy in that simulator. However, the simulator can be inaccurate or totally off the real environment. For example, the driving environment in New York is different with that in Alabama. Therefore, in the second setup of assumption, we assume having a distribution of environments. The second setup of assumption will be elaborate at the end of this section.

For the first setup, we assume that:

1. The simulator for training and testing the decision-making policy is accurate.
2. The AV is surrounded by only Human-controlled Vehicles (HVs).
3. We do not have a human driver behavior model. Instead, we can only get information from observations from the simulations.
4. The perception and actuation subsystems of the AV are perfect.

Under these assumptions, the objective of the synthesis problem is to design an optimal decision-making subsystem using reinforcement learning. To find an optimal policy, one needs to tackle the exploration-exploitation trade-off of reinforcement learning. This will be elaborated in Chapter 2.

No matter how carefully we design the simulator, it cannot represent the real world totally accurately. There are many reasons. First, the lack of data. We can collect as much data as we want, but it is never complete. For example, right now, all surrounding vehicles are human-controlled vehicles (HVs). While in the future, AV may interact with both human drivers and other AVs. Moreover, we cannot enumerate all the environments in various locations, which will

also require tremendous amounts of data. Therefore, it is necessary to study the synthesis problem under the assumption that the **environment model or the simulator is not accurate**. Regarding the above-mentioned situation, we assume that:

1. The simulator for training and testing the decision-making policy is not accurate.
2. We are not training the decision-making policy using only one specific simulator. We are training the policy in the **distribution of simulators**.
3. We do not have a model of human driver behaviors. We can only get information from observations and infer human driver behaviors indirectly.
4. The perception and actuation subsystems of the AV are perfect.

As may have already been noticed, a new assumption was added to require the probability distribution of these simulators, which requires more data. The objective of the synthesis problem under the second set of assumptions is to train a decision-making policy that can fast adapt to different environments. Therefore, the distribution of simulators is used to train the adaptation ability of the policy. So, the distribution does not need to be very accurate, just enough to cover a range of possibilities. After training, we hypothesize that the policy will have the ability to adapt to different environments more quickly given a small quantity of data. The second synthesis problem is elaborated in Chapter 5.

1.4 AV Evaluation Problem

The objective of the evaluation problem studied in this work is to evaluate the performance of the AV control system. As shown in Figure 1.4, the typical hierarchical architecture of AVs includes the perception system (where TSD denotes Traffic Signalization Detection and MOT denotes Moving Objects Tracking) and the decision-making system. In this work, we do not discuss any perception-related problem. We assume a perfect perception system is available.

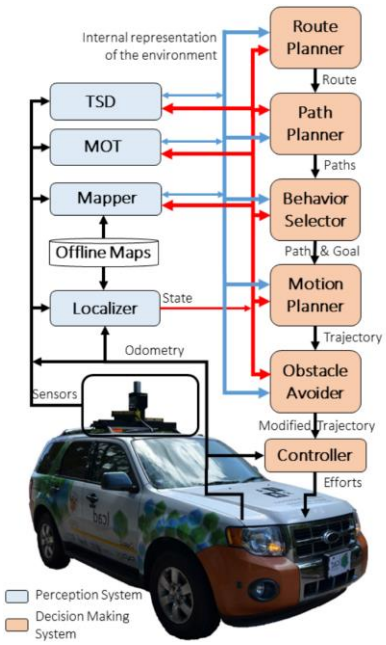


Figure 1.4 Overview of the Typical Hierarchical Architecture of AVs [12]

In this dissertation proposal, we study the evaluation problem under two sets of assumptions. For the first setup, we assume that:

1. The testing AV is surrounded by only Human-controlled Vehicles (HVs).
2. We have access to a large amount of naturalistic driving data, which captures the behavior of human drivers.
3. We do not have access to the control system of the testing AV, which means we cannot calculate the AV's state (e.g., the position and velocity) directly from a model. We can only observe the state, i.e., it is black-box testing.
4. We do not know anything about the "danger region" of the model, which means we cannot "skew" the distribution directly.

The objective of the evaluation problem under the assumptions above is to estimate the crash rate and in general, the safety performance of an AV system accurately and efficiently. More specifically, the goal is to develop a new accelerated evaluation method that can overcome the *curse of dimensionality* issue and perform "black-box" testing (by searching the state space) with a comparable accelerated rate achieved by the state-of-the-art methods.

Moreover, we will consider another set of assumptions when naturalistic driving data is not available. Usually, even when a large amount of data is collected, they are from a limited

region or country. The driver behaviors in different cities or countries can be very different. This is a common problem for any data-driven control systems that assume "data is available." In addition, there will be more and more AVs and Connected Vehicles (CVs) on the road in the future, and consequently, the environmental statistics will change with time. Therefore, simply relying on one fixed database to evaluate AVs is not good enough. Due to these reasons, we have a second set of assumptions:

1. We do not have access to a large quantity of naturalistic driving data.
2. We do not have access to the control system of the AV, and thus, we perform black-box testing.
3. We have access to the AV's output and thus can conduct searching and learning.
4. We assume this work will be performed only under computer simulations or hardware-in-the-loop simulations since real-world testing is too slow and expensive.

The objective of the evaluation problem under the second set of assumptions is to find the "danger region" in the state space, meaning that we want to find a time-series of inputs from the environment which will lure the AV to a crash that is the responsibility of the AV.

1.5 Contributions

In this dissertation, we proposed new synthesis approaches and new evaluation approaches and make contributions in the following three aspects:

- We developed a reinforcement-learning-based method to solve the synthesis problem and designed a discretionary lane change policy that helps the AV travel safely and efficiently on a highway.
- To evaluate this discretionary lane change policy and its low-level safeguard, we developed two evaluation approaches.
    - We extended the accelerated evaluation method introduced in [5] and developed a novel approach to accelerate the evaluation further and to test the low-level safeguard system in a more complex environment.
    - We developed an evaluation method that **does not need environmental statistics**. The approach can generate socially acceptable attacks that can lure the AV to AV-responsible crashes.

- Given the evaluation results and socially acceptable attacks, we use meta reinforcement learning to design an adaptive policy that can quickly adapt to different attacks from the environments. It results in a low crash rate in all situations.

1.6 Outline of the Dissertation

The remainder of this dissertation is organized as follows. In Chapter 2, we designed a discretionary lane change decision-making policy using the reinforcement learning method. In Chapter 3, the lower level safeguard system of the prior designed policy is evaluated with an advanced accelerated evaluation method named subset simulation. In Chapter 4, the whole policy is evaluated by designing a socially acceptable perturbation that keeps challenging the learned policy. In Chapter 5, we developed an adaptive lane change decision-making policy that has robust performance when facing attackers and other different driving conditions. Finally, we discuss future research directions and conclude our work in Chapter 6.

Chapter 2 Synthesis of the Autonomous Vehicle's Policy Using Reinforcement Learning

One of the major synthesis problems for AVs is to design a reliable decision-making subsystem. In this chapter, we will study this synthesis problem under the first set of assumptions described in Section 1.3 .where we assume a perfect simulator, and we try to solve the discretionary lane change decision-making problem. We use the Markov Decision Processes (MDP) framework to represent the randomness in the environment from other road users. We solve the MDP by using the Reinforcement Learning (RL) approach discussed in Section 1.2.1 . The RL agent needs to deal with the tradeoff between exploitation and exploration. In this chapter, we develop a **new model-based exploration approach** that guides the agent to explore the state space more efficiently.

2.1 Literature Reviews on Discretionary Lane Change Decision-making Approaches

Discretionary Lane Change (DLC) usually happens when a driver wants to drive faster, keep a greater following distance, have a better line of sight, maintain better ride quality, etc. The AV needs to solve a decision-making problem considering multiple objectives by utilizing the surrounding vehicles' information for efficient and safe operation.

Recently, the DLC decision-making problem has been researched using a variety of approaches. In [14], Kesting et al. developed a rule-based general lane change strategy called Mobil that depends on the total acceleration potential gain or loss of the ego vehicle and surrounding vehicles. The acceleration potential of each vehicle is calculated from the Intelligent Driver Model (IDM). However, this method requires a politeness parameter and a threshold parameter beforehand, and therefore, it depends on human experience or data to tune the parameters. In [77], the authors modeled the lane change behavior using the game theory under Vehicle-to-Vehicle (V2V) circumstance by finding the Nash equilibrium of two interacting vehicles. The game theory approaches are widely used in computer game applications. However, it needs to model multiple agents' intentions, which is not feasible under our assumptions.

Another method for solving the DLC decision-making problem is Deep Reinforcement Learning (DRL). The state-of-the-art DRL techniques have been proven to be useful for automatically learning policies for difficult decision-making problems like winning the Go game [78]. Unlike traditional controller design methods, reinforcement learning can generate control policies without relying on explicit system dynamics. Several published papers demonstrate DRL's ability to solve the DLC decision-making problems [24], [79]. However, the trained policy still suffers from the exploration problem. Currently, the researchers are using rule-based exploration approaches, which correspond to the reward function [23], and this method goes against the assumption that the MDP is unknown. Therefore, it is necessary to develop a generic and versatile exploration method.

2.2 Reinforcement Learning Fundamentals

In this section, the background knowledge of reinforcement learning is introduced. In reinforcement learning, an agent tries to learn an optimal policy to maximize the future discounted cumulative reward by directly interacting with the environment. The problem is first modeled as a Markov decision process (MDP), which is defined as a 5-tuple: $M = (S, A, P, r, \gamma)$, where $S \subseteq R^n$ is the state, $A \subseteq R^m$ is the action, and $P: S \times A \to \Delta(S)$ is the stochastic transition dynamics, where $\Delta(S)$ is a probability distribution on S. $r: S \times A \to R$ is the reward function, while $\gamma \in (0,1]$ is the discount factor. The discounted cumulative reward is $r_{0:t} = \sum_{t=0}^{\infty} \gamma^t r_t$, where r_t is the reward at time t.

For each time step t, the agent is trying to learn a policy $\pi_\alpha(s_t) = a_t$ with parameters α, where $s_t \in S$ is the state at time t and $a_t \in A$ is the action at time t. The expectation of future discounted cumulative reward starting from state s following policy π_α can be described as:

$$V(s|\pi_\alpha) = E_{\pi_\alpha, M}\left[\sum_{t=0}^{\infty} \gamma^t r_t \ |s_0 = s\right] \tag{2.1}$$

where V is the value function. And the action-value function Q is defined as:

$$Q(s, a|\pi_\alpha) = E_{\pi_\alpha, M}\left[\sum_{t=0}^{\infty} \gamma^t r_t \ |s_0 = s, a_0 = a\right] \tag{2.2}$$

The objective of RL is to maximize the expected discounted cumulative reward, i.e., $E_{\pi,M}[\sum_{t=0}^{\infty}\gamma^t r_t]$. For problems with discrete action space, we can get the optimal policy by learning an accurate Q function Q^* and thus we have $\pi(s) = argmax_a Q^*(s,a)$.

2.2.1 Dynamic Programming

The optimal policy for an MDP can be found by using Dynamic Programming (DP) when we know the MDP's identities (i.e., the transition model, reward function, etc.) Given the complete model, we can write the Bellman optimality equations, break the problem into subproblems and then solve them. There are two ways of using DP to solve MDP: policy iteration and value iteration.

The algorithm first starts with a random policy. The policy iteration method consists of two steps. The first step is the policy evaluation step, and the second step is the policy improvement step. During the policy evaluation step, we find the value function of that policy. And then, during the policy improvement step, the policy is improved based on the previously learned value function. In this process, each policy is proved mathematically to be a strict improvement over the previous one. We have summarized the policy iteration algorithm in Algorithm 2.1.

Algorithm 2.1: Policy Iteration Algorithm

Initialization: $V(s) \in R$ and $\pi(s) \in A$ are initialized arbitrarily for all $s \in S$

\# Policy Evaluation:
while $\Delta > \theta$ **do**
    Set $\Delta = 0$
    for $s \in S$ **do**
        Set $temp = V(s)$
        $V(s) = \sum_{s'} p(s'|s,\pi(s))[r(s,s',\pi(s)) + \gamma V(s')]$, where s' is the next state
        $\Delta =$ max$(\Delta, |temp - V(s)|)$

\# Policy Improvement:
Set $stable = true$
for $s \in S$ **do**
    Set $temp = \pi(s)$
    $\pi(s) = argmax_a \sum_{s'} p(s'|s,a)[r(s,s',a) + \gamma V(s')]$, where s' is the next state
    if $temp \neq \pi(s)$: set $stable = false$
    if $stable = true$: **terminate**; **else**: go to # Policy Evaluation.

We start with a random policy and value function for value iteration methods, then using the collected data to find an improved value function iteratively until reaching the optimal value function. The value iteration method is the result of directly applying the optimal Bellman operator to the value function in a recursive manner. After finding the optimal value function, the optimal policy can be easily derived from it. The value iteration algorithm is summarized in Algorithm 2.2.

Algorithm 2.2: Value Iteration Algorithm

Initialization: $V(s) \in R$ and $\pi(s) \in A$ are initialized arbitrarily for all $s \in S$

# Value Iteration:
while $\Delta > \theta$ **do**
    Set $\Delta = 0$
    for $s \in S$ **do**
        Set temp = $V(s)$
        $V(s) = \max_a \sum_{s'} p(s'|s,a)[r(s,s',a) + \gamma V(s')]$, where s' is the next state
        $\Delta = \max(\Delta, |temp - V(s)|)$

Output: a deterministic policy $\pi(s) = argmax_a \sum_{s'} p(s'|s,a)[r(s,s',a) + \gamma V(s')]$

Both algorithms work theoretically, and there is no significant difference between the policy iteration and the value iteration algorithms. The value iteration is simpler, but it is computationally heavy, while the policy iteration is simpler, and it is relatively computationally cheap. However, they require the MDP transition probability function and reward function, which is not feasible in our application.

2.2.2 Temporal Difference Method

Different from the DP methods, the Temporal Difference (TD) method is model-free, which does not require any information of the MDP. The TD method learns the optimal policy by bootstrapping from the current estimation. TD method can be used to learn both the value function and the Q function. The simplest TD method is the State–action–reward–state–action (SARSA) algorithm. SARSA learns by interacting with the environment and collect the 5-tuple (state, action, reward, next state, next action), i.e. (s, a, r, s', a'), and then update the Q function based on the TD method. The SARSA algorithm is summarized in Algorithm 2.3.

Algorithm 2.3: SARSA Algorithm

Initialization: $Q(s,a) \in R$ is initialized arbitrarily for all $s \in S$

# Iteration to estimate the Q function:
for each episode **do**
    Choose the action a given the state s using policy derived from Q (ϵ-greedy)
    for each timestep of this episode **do**
        Apply the action a and observe the reward r and the next state s'
        Choose the next action a' given s' using policy derived from Q (ϵ-greedy)
        $Q(s,a) = Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$, where α is the learning rate
        Set $s = s'$; $a = a'$

Output: a deterministic policy $\pi(s) = argmax_a Q(s,a)$

Different from the SARSA algorithm, Q learning updates the Q function using the maximum Q over all possible actions for the next step. i.e., $Q(s,a) = Q(s,a) + \alpha[r + \gamma max_a Q(s',a) - Q(s,a)]$. Since the action used for updating the policy ($a = argmax_a Q(s',a)$) is different from the action being taken at state s' (with ϵ possibility to be a random action), the Q learning method is an off-policy RL method. While the SARSA updates the policy using the action that will be taken at state s', then the SARSA is an on-policy RL method.

Algorithm 2.4: DQN Algorithm

Initialization:
    $Q(s,a;\beta) \in R$ network is initialized with random parameter β
    Target Q function $\hat{Q}(s,a;\beta^-)$ is initialized with parameter $\beta^- = \beta$
    Initialize replay buffer D with capacity N to store past experience

# Iteration for estimating Q function:
for each episode **do**
    Initialized the episode with a random starting state
    for each timestep of this episode **do**
        Choose the action a given the state s using policy derived from Q (ϵ-greedy)
        Apply the action a and observe the reward r and the next state s'
        Store the tuple (s,a,r,s') in D
        Sample random minibatch of tuples (s,a,r,s') from D
        Set target Q value: $y = r + \gamma max_a \hat{Q}(s',a;\beta^-)$
        Perform a gradient descent step on $(y - Q(s,a;\beta))^2$ for β
        Every C steps, reset $\hat{Q} = Q$, i.e. $\beta^- = \beta$

Output: a deterministic policy $\pi(s) = argmax_a Q(s,a)$

To solve a high dimensional or continuous state space MDP problem, a functional approximation (e.g., a neural network that approximates Q function) becomes necessary to ensure the solution's tractability. However, directly implementing the Q-learning method for continuous state space MPD problem with approximation could cause the Q function to diverge. The Deep Q-Network (DQN) method in [80] has successfully demonstrated its value function convergence with empirical results using techniques such as "experience replay" and "periodically updated target network." The algorithm is summarized in Algorithm 2.4.

There are many extensions of DQN to improve the training performance, such as dueling DQN [81], which estimates the value function and advantage function ($A(s,a) = Q(s,a) - V(s)$) with shared network parameters. And the Double DQN (DDQN) [82], which estimate the target Q value by $y = r + \gamma\hat{Q}(s', argmax_a Q(s', a; \beta); \beta^-)$. DDQN method uses the Q network rather than the target Q network to choose the next step action for estimating the Q value. DDQN shows benefits in reducing overestimating Q value with very little computational burden.

Here, we will also briefly introduce the Monte Carlo (MC) method that can also be used to solve the MDP problem. MC method does not belong to the category of TD methods. Instead of using the Bellman equation to estimate the value function or the Q function, MC directly calculates the true value for each episode and then updates the policy by $V(s_t) = V(s_t) + \alpha(G(s_t) - V(s_t))$, where $G(s_t) = \sum_{k=t}^{T} \gamma^{k-t} r_{k-t}$ is called the empirical return, which is calculated from a complete episode. The difference between DP, MC, and TD methods are summarized in Figure 2.1.

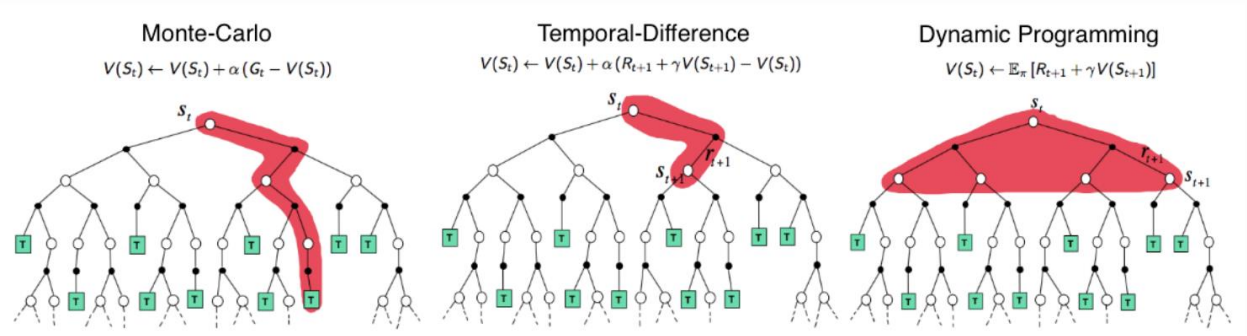


Figure 2.1 Comparison of the backup diagrams of Monte-Carlo, Temporal-Difference learning, and Dynamic Programming for state value functions [83]

2.2.3 Policy Gradient Methods

Most methods mentioned above aim to learn the value function or Q function first, and then the select actions. The Policy Gradient (PG) methods learn the policy directly with a parameterized function with respect to θ, i.e., $\pi(s;\theta) \in A$. The policy is updated by gradient ascend with respect to the objective function:

$$J(\beta) = \sum_{s \in S} d^{\pi}(s) \sum_{a \in A} \pi_{\theta}(s) Q^{\pi}(s,a) \tag{2.3}$$

where the $d^{\pi}(s)$ is the stationary distribution of this MDP under the policy $\pi_{\theta}(s)$. To maximize the objective method, researchers have developed a variety of algorithms. Here we will not expand them into details. The simplest policy gradient method is the REINFORCE algorithm, which relies on an estimated return by Monte Carlo methods using episode samples (i.e., the $G(s)$) to update the policy parameter θ via gradient ascent [84]. However, the vanilla REINFORCE algorithm suffers from noisy gradients and high variance problems [85], which contribute to the instability and slow convergence of the REINFORCE method.

To improve the policy gradient method, researchers use other estimation of the cumulative return instead of the sampled episodes. In Figure 2.2, we summarize the classic variants of policy gradient methods. Except for the REINFORCE algorithm, in other methods shown in Figure 2.2, researchers introduced some type of the "value function" in addition to the policy, which is proven to be useful in reducing the instability of the gradient [86]. That will result in having two networks, one for the policy (which is called the Actor) and one for the "value function" (which is called the Critic). Therefore, these methods are also called Actor-Critic (AC) methods. Specifically, researchers use the Q function [87] (Q Actor-Critic), advantage function [87](Advantage Actor-Critic) and TD function [87] (TD Actor-Critic) as objective function.

$$\begin{aligned}
\nabla_{\theta} J(\theta) &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s,a)\, G_t] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s,a)\; Q^{w}(s,a)] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s,a)\; A^{w}(s,a)] && \text{Advantage Actor-Critic} \\
&= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s,a)\; \delta] && \text{TD Actor-Critic}
\end{aligned}$$

Figure 2.2 Different Types of Objective Functions for the Policy Gradient Methods [86]

It is natural to expect the policy gradient-based methods are more useful in continuous action space. Because there is an infinite number of actions to estimate the values for and hence the value-based approaches are way too expensive computationally. However, for discrete action space, using two networks (the actor network and the critic network) is not necessarily better than the TD methods (e.g., DQN and DDQN). Moreover, policy gradient methods require more training loops and generally less data-efficient than the TD methods. Therefore, if the action space of the MDP is discrete, TD methods will perform better. Since our application has a discrete action space, the backbone RL method we implement is the **DDQN** method.

2.2.4 Exploration-exploitation Trade-off and Current Methods

Exploration-exploitation trade-off is a critical topic in reinforcement learning. We want the RL agent to learn the best policy as fast as possible. However, the agent has no data to play with at the beginning. It needs to explore the state space (i.e., the environment) and collect data to update the policy. If the collected data are bad experiences, it could lead to local minima or total failure. The deep RL algorithms use a neural network that optimizes for the best returns can achieve exploitation quite efficiently, while exploration remains an open topic.

When solving the DLC decision-making problem, studies [24], [88] mainly focus on building the environment simulator and then use a simple ε-greedy exploration strategy. However, these papers' backbone RL methods still struggled with the exploration-exploitation challenge and provided only heuristic exploration approaches. In [24], the authors stated that the standard DRL approach might require a lot of samples before learning the situations when action would lead to a collision, thus leading to learning inefficiency. Therefore, they implemented human-designed **short-horizon safety checks** to guide the exploration by eliminating unsafe actions in certain conditions. Similarly, in [88], the authors developed a DRL method with rule-based constraints. These exploration methods are heavily guided by human design, which is contradictory to the spirit of deep learning (to learn without the human-designed feature), or full exploration (without presumptions or crutches). Therefore, an advanced exploration method is needed.

Previous methods are reviewed here first to find a systematic exploration technique for the DLC decision-making application. From the beginning of reinforcement learning history, researchers started to realize the importance of exploration. Thrun [89] and Kaelbling [90]

developed an exploration strategy for the agent to choose its action based on some fixed distributions randomly. These methods are now known as the undirected exploration strategy. However, undirected exploration strategies are not efficient. Therefore Thrun developed a counter-based and a recency-based method that the agent can explore based on how many times it has visited different states [91]. Kolter extended the idea by setting the initial reward of all states to a high value, which will decrease during exploration in [92]. In [93], Brafman introduced the R-max method, where the transition environment model is built and updated by counter-based results. These methods are now known as the directed exploration strategy. However, for high-dimensional environments, it is inefficient to explore the whole state space, and it is hard to implement these methods in continuous state environments. Motivated to overcome these problems, Achiam introduced surprise-based intrinsic reward in [94], where the agent is excited to see outcomes that run contrary to its understanding of the world. Also, in [95], the authors first map states to feature space and then define the difference between the estimated feature and the true feature as the curiosity intrinsic reward. In [96], the authors use variational auto-encoder (VAE) to build the environment model and help with exploration. These methods are known as model-based exploration because they all explore based on environment model. However, in these papers, the authors did not focus on how to build the environment model. Instead they focus on how to construct the intrinsic reward. Therefore, in this chapter, we will develop a better exploration method that can help finding the optimal policy.

2.3 Model-based Exploration of Reinforcement Learning

The objective of this section is to develop a method to learn and use the environmental model to guide the exploration of the agent to "unfamiliar" states. The agent would form a concept of "the world" around it and intrinsically seek outcomes that run contrary to, or deviating from, its understanding of the world. Therefore, we propose a model-based exploration method as shown in Figure 2.3. Four key components are needed for our implementation task. First, the reinforcement learning backbone Double Deep Q Network (DDQN); second, building an environment model; third, define intrinsic reward for exploration based on the model; and finally, the model updating and policy learning strategy.

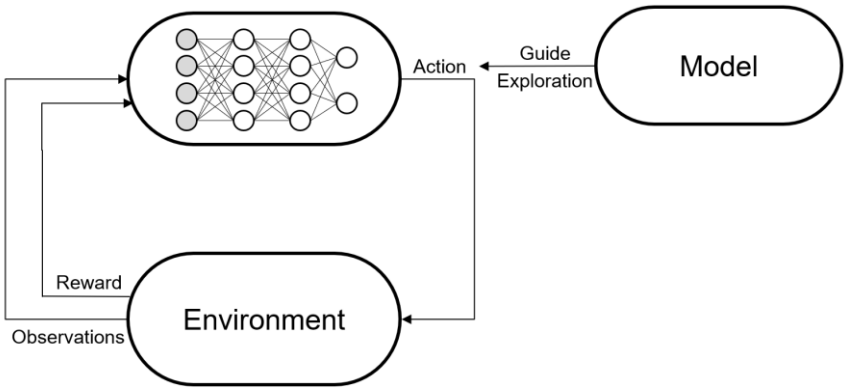


Figure 2.3 Model-based Exploration

2.3.1 Simulator and Reward Function

In this section, the state space and the action space of the discretionary lane change decision-making problem, the objective related original reward (without the intrinsic reward modification), and the simulation environment are introduced. For benchmark purposes, the problem definition and the simulation environment are the same as the one used in [24].

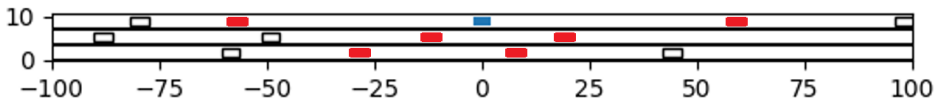


Figure 2.4 Three Lane Highway Simulator.

The simulator used in this work is a three-lane highway simulator based on [24]. The host vehicle is driving with the information of the surrounding six nearest vehicles (three vehicles in front, three vehicles behind) as shown in Figure 2.4. The blue box is the host vehicle, and the six red boxes are the nearest surrounding vehicles whose states are observed. The remaining boxes are environment vehicles whose states are not observed. The surrounding vehicles' driving strategy is also described in [24].

The state-space $S \subseteq R^n$ of the learning agent (host vehicle) includes the host vehicle's lateral position y, host vehicle's longitudinal velocity v_x and the relative longitudinal position of the i^{th} surrounding vehicle Δx^i, and the relative lateral position of the i^{th} surrounding vehicle

Δy^i and the relative longitudinal velocity of the i^{th} surrounding vehicle Δv_x^i. Therefore, in total, we have a continuous state space of $2 + 3 \times 6(\text{cars}) = 20$ dimensions, i.e., $S \subseteq R^{20}$.

The actions of both the host vehicle and surrounding vehicles are discrete. As defined in [24], we consider four action choices along the longitudinal direction a_x, namely, *maintain speed*, *accelerate*, *brake*, and *hard brake*. Whereas for lateral direction actions a_y, we assume three choices, *lane keep*, *change lane to right*, and *change lane to left*. In total, we define 12 different discrete actions $a = [a_x, a_y]$.

The original reward from the environment r^e (not considering intrinsic reward) is defined as in [24]. It is formulated as a function of (dx, y, v_x), where dx is the distance between the host vehicle and its lead vehicle, y is the lateral position of the host vehicle and v_x is the longitudinal velocity of the host vehicle. The reward is defined as follows.

$$r_x = \begin{cases} \exp\left(-\frac{\left(dx - dx_{safe}\right)^2}{10dx_{safe}}\right) - 1 & \text{if } dx < dx_{safe} \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

$$r_y = \exp\left(-\frac{(y - y_{des})^2}{y_{norm}}\right) - 1 \tag{2.5}$$

$$r_v = \exp\left(-\frac{(v_x - v_{des})^2}{v_{norm}}\right) - 1 \tag{2.6}$$

where the dx_{safe}, y_{des} and v_{des} are the safe longitudinal distance to the lead vehicle, the target lane position, and desired speed, respectively. These three rewards are normalized by $10dx_{safe}$, y_{norm} and v_{norm}, respectively, so that no reward dominates the total reward. Then we have $r^e = \frac{1}{3}(r_x + r_y + r_v)$ with no collision and $r^e = -2$ if there is a collision.

2.3.2 Environment model

In this section, we developed an environment model that will be used for deriving intrinsic reward (Section 2.3.3). As defined in Section, we have a 20-dimension state-space S. To take advantage of general application domain background knowledge (vehicle kinematics), the model of environmental vehicles is factorized into two parts, the deterministic vehicle

kinematics model and the statistical human behavior model. Given the action $a^i = [a_x^i, a_y^i]$ of the i^{th} surrounding vehicle, the action $a = [a_x, a_y]$ of the host vehicle and the current state s, the next state s' is deterministically derived from the vehicle kinematics model:

$$\text{Host car: } \begin{cases} y' = y + a_y \Delta t \\ v_x' = v_x + a_x \Delta t \end{cases} \tag{2.7}$$

$$i^{\text{th}} \text{ car: } \begin{cases} (\Delta x^i)' = \Delta x^i + \Delta v_x^i \Delta t + \frac{1}{2}(a_x^i - a_x)\Delta t^2 \\ (\Delta y^i)' = \Delta y^i + (a_y^i - a_y)\Delta t \\ (\Delta v_x^i)' = \Delta v_x^i + (a_x^i - a_x)\Delta t \end{cases} \tag{2.8}$$

The question now is how to infer the ai from the human behavior model. Traditional human driver models usually have fixed structures based on various assumptions and do not best represent driver behavior when the scenario varies [14], [97]. In our application, it is best to learn a model that can infer a wide range of feasible actions with their corresponding probabilities. Therefore, in this work, we choose the Variational Auto-Encoder (VAE) to represent human behavior.

The VAE is typically used for learning latent representation z in an unsupervised manner. Because of the fact that any distribution can be generated by mapping a normal distribution through a sufficiently complicated function, the distribution of z is asserted to be standard normal distribution $N(0, I)$. An auto-encoder network is actually a pair of two connected networks, an encoder q, and a decoder p. An encoder network q takes in an input and converts it into a smaller, dense representation, which the decoder network p can convert it back to the original input. VAEs have already shown promise in representing many kinds of complicated data, as in [98]. In our application, we take the current state s and the action a of the host vehicle as an input of the model and predict the next state s', therefore we build the VAE to condition on $[s, a]$, which is called Conditional Variational Auto-Encoder (CVAE) [99].

The whole model structure is shown in Figure 2.5. Given the current state s and the host vehicle's action a, the predicted action of the i^{th} environment vehicle $\widehat{a^i}$ is estimated from the decoder of CVAE. Then both $\widehat{a^i}$ and $[s, a]$ is given to the vehicle kinematics model, and the next state $\widehat{s'}$ is estimated. The CVAE is trained with the next state s' as the target value. If the observed surrounding vehicles change in one time-step, there will a sudden jump in

corresponding dimensions of the next state s'. To solve this problem, we remap these id-changed surrounding vehicles' corresponding state to correct position according to the expected action $\hat{a}$ and set newcomer or disappeared vehicles' state to the default value.

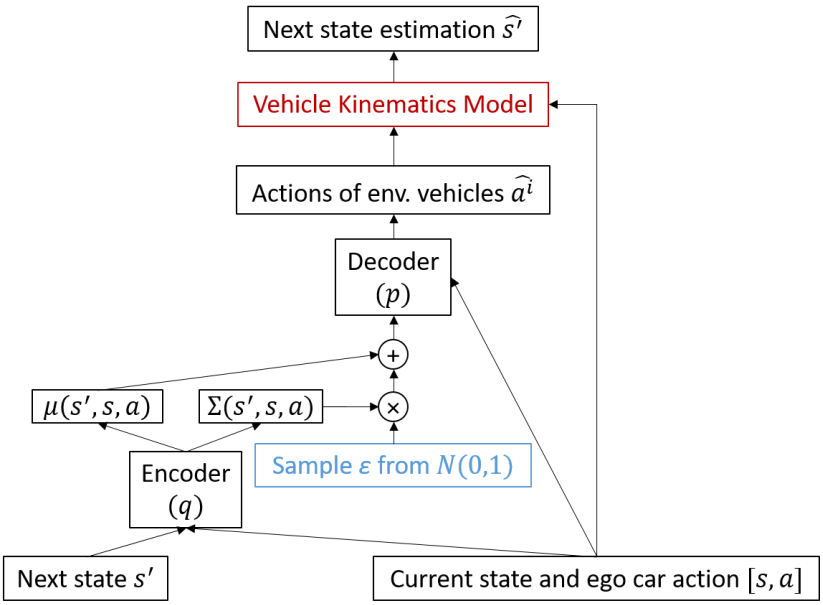


Figure 2.5 Conditional variational auto-encoder model with the vehicle kinematics model

The loss function for training a CVAE is derived from the objective function that aims to guess the next state best. Formally speaking, the objective function is to maximize the conditional log-likelihood $\log p_\theta(s'|s,a)$ with parameter θ. As shown in [99], this objective function is normally intractable, so the authors apply the stochastic gradient variational Bayes (SGVB) framework to train a statistical aggregated model. Therefore, the variational lower bound (also called Evidence Lower BOund, i.e., ELBO) of the log-likelihood is used as the objective function. In our application, conditional log-likelihood is written as:

$$\begin{aligned} &\log p_\theta(s'|s,a) \\ &\quad = KL(q_\phi(z|s',s,a)||p_\theta(z|s',s,a)) \\ &\quad + E_{q_\phi(z|s',s,a)}\left[-\log q_\phi(z|s',s,a) + \log p_\theta(z,s',s,a)\right] \end{aligned} \tag{2.9}$$

where p_θ is the decoder with parameter θ, q_ϕ is the encoder with parameter ϕ and $KL(\cdot\,||\,\cdot)$ is the KL-divergence function. And the evidence lower bound (ELBO) can be derived as:

$$\log p_\theta(s'|s,a) \geq -KL(q_\phi(z|s',,a)||p_\theta(z|s,a)) + E_{q_\phi(z|s',s,a)}[\log p_\theta(s'|z,s,a)] \quad (2.10)$$

Recall that $p_\theta(z|s,a) \sim N(0,I)$ and $q_\phi(z|s',s,a) \sim N\big(\mu(s',s,a),\Sigma(s',s,a)\big)$ denoted as $N(\mu,\Sigma)$, the first term on the right-hand side of Equation (2.10) can be computed in closed form. For the second term, using the reparameterization trick described in [100], we can have the empirical lower bound as:

$$\widehat{L_{CVAE}} := -\frac{1}{2}(\mathrm{tr}(\Sigma) + \mu^T\mu - k - \log\det(\Sigma)) + \frac{1}{L}\sum_{l=1}^{L}\log p_\theta\,(s'|z^l,s,a) \quad (2.11)$$

where k is the dimension of the distribution, z^l is a sample from $q_\phi(z|s',s,a)$ and L is the total number of samples. In our application, the predicted next states are continuous. Therefore the $p_\theta(s'|z,s,a)$ is modeled by an i.i.d. Gaussian distribution, i.e. $p_\theta(s'|z,s,a) \sim N(f(z,s,a),\sigma^2 I)$, where $f(z,s,a)$ is the output of the decoder and σ is a hyperparameter controlling how precise the model is. In more detail, from [98], $\log p_\theta\,(s'|z^l,s,a) = -\frac{n}{2}\log 2\pi\sigma^2 - \frac{1}{2}\|s' - f(z^l,s,a)\|^2/\sigma^2$. We could sample $z^l \sim q_\phi(z|s',s,a)$ to estimate it, but getting a good estimate would require passing many samples of z^l through $f(z^l,s,a)$, which would be inefficient. Therefore, as is standard in stochastic gradient descent, we take one sample of $z \sim q_\phi(z|s',s,a)$ to approximate $\frac{1}{L}\sum_{l=1}^{L}\log p_\theta\,(s'|z^l,s,a)$. In summary, we have:

$$\log p_\theta\,(s'|s,a) \geq \mathrm{ELBO} \approx \widehat{L_{CVAE}} \quad (2.12)$$

2.3.3 Intrinsic reward

In this section, we derive the surprise-based intrinsic reward from the CVAE model shown in Section 2.3.2 to help the agent explore states that run contrary to its understanding of the world. As defined in [94], the exploration incentive is:

$$\eta\, E_{s,a\sim\pi}\,[KL(p(s'|s,a)||p_\theta(s'|s,a))] \quad (2.13)$$

where η is a weighting factor, and p represents the ground truth environment model. It is the on-policy average KL-divergence of p_θ from p and is intended to measure the host vehicle's surprise about the observation s'. The model p_θ, corresponding to the decoder in the CVAE model (with vehicle kinematics model), should only be close to p in regions of the state space that the host

vehicle has already observed and captured the transition rule. Therefore, the KL-divergence of p_θ and p will be higher in unfamiliar places. Essentially, this encourages the agent to go where it has not been modeled correctly, which in term implies the places it is unfamiliar. As derived in [94], adding the surprise exploration incentive in Equation (2.13) gives the net effect of performing a reward shaping of the form:

$$r'(s', s, a) = r^e + \eta\big(\log p\,(s'|s, a) - \log p_\theta\,(s'|s, a)\big) \tag{2.14}$$

where r' is the new reward, r^e is the original environment reward defined in Section 2.3.1 . Practically, the ground truth environment transition model p is unknown. Therefore, they use the cross-entropy approximation of the $KL(p||p_\theta)[s, a]$ instead:

$$KL(p||p_\theta)[s, a] = H(p, p_\theta)[s, a] - H(p)[s, a] \approx H(p, p_\theta)[s, a] \tag{2.15}$$

where $H(p, p_\theta)[s, a] \coloneqq E_{s'\sim p}[-\log p_\theta\,(s'|s, a)]$ is the cross-entropy for distributions p and p_θ. For a detailed explanation, please refer to [94]. Then, this approximation results in a reward shaping of the form:

$$r'(s', s, a) = r^e - \eta \log p_\theta\,(s'|s, a) \tag{2.16}$$

From Equation (2.12), we have derived the lower boundary of conditional log-likelihood $\log p_\theta\,(s'|s, a)$. Therefore, the upper boundary of the second term in Equation (2.16) is $-\eta\widehat{L_{CVAE}}$. Also, as shown in [98], the difference between $\widehat{L_{CVAE}}$ and $\log p_\theta\,(s'|s, a)$ will vanished as the model becomes more and more accurate. Therefore, we use the upper boundary as the surprise-based intrinsic reward denoted as r^{sp}, i.e.:

$$r^{sp} \coloneqq -\eta\widehat{L_{CVAE}} \tag{2.17}$$

$$r'(s', s, a) = r^e + r^{sp} \tag{2.18}$$

Similar to [94], we keep the average intrinsic reward r^{sp} upper-bounded by adjusting η at each iteration:

$$\eta = \frac{\eta_0}{\max\left(1, \frac{1}{|B|}\sum_{(s,a,s')\in B} r^{sp}\right)} \tag{2.19}$$

where B is the batch of data used for the policy update step. For every batch of tuple data (s_t, a, s_{t+1}, r^e), the r_t^{sp} is calculated from the current CVAE model, as shown in Figure 2.6.

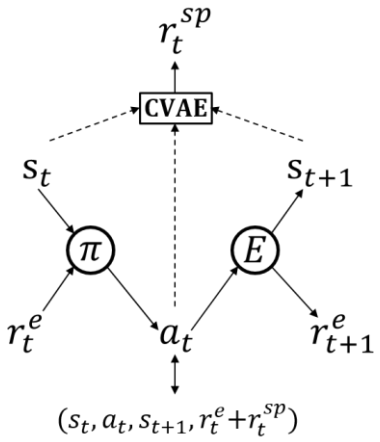


Figure 2.6 Intrinsic Reward

2.3.4 Model and policy update strategy

The final algorithm is summarized in Figure 2.7. Same as traditional DDQN algorithm, after initialization, samples (s, a, s', r^e) from the environment are collected by the agent following its current policy and stored in the replay buffer. During training the DDQN, a batch of samples is sent to CVAE to calculate r^{sp}, then used for updating the DDQN policy network. Also, we update the CVAE model every 100 episodes with samples (s, a, s') from the replay buffer. The expected outcome is that the CVAE model becomes more and more accurate, and we have a well-explored DDQN policy. Finally, at the end of the training, the intrinsic reward should vanish, and the learned policy should be optimized corresponding to environment reward r^e only.

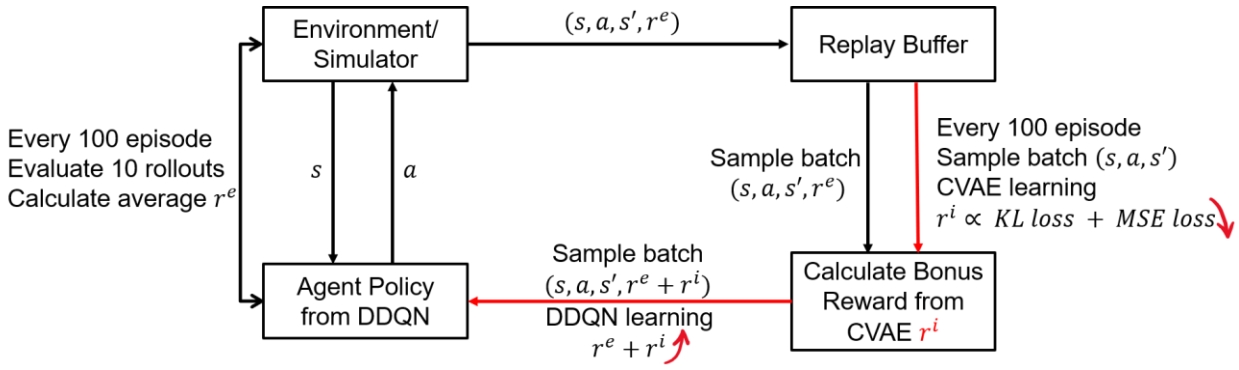


Figure 2.7 Training procedure for CVAE and DDQN

2.3.5 Baseline Exploration Method

The proposed model-based exploration method is compared with annealing ε-greedy exploration and rule-based safety check exploration. In the annealing ε-greedy exploration, in which the agent chooses an action based on:

$$\pi(a|s) = \begin{cases} 1-\varepsilon, & \text{if } a = argmax_a Q(s,a) \\ \varepsilon, & \text{random action} \end{cases} \tag{2.20}$$

where the ε is a small probability on which the agent will choose random action. It decreases with the training process as $\varepsilon = \max\left(e^{-iC}, \varepsilon_0\right)$, where i is the training episode, and C is the annealing factor.

Meanwhile, the short-horizon safety check baseline method will replace dangerous action before applying it. If the action chosen by the DDQN is unsafe, it will be replaced by Intelligent Driver Model (IDM) and Advanced Emergency Braking (AEB) system based action. The key safety check includes:

For longitudinal actions without changing lane:

- Check the safe distance and time to collision (TTC) with the leading vehicle. If the action violates the safety check, it will be replaced by a safe longitudinal action.

For lane change action:

- If the ego vehicle is in the left-most lane, change to the left is not valid, similar for the right lane.

- The safe distance and TTC to the target lane vehicle are continuously monitored. If it is violated, the lane change action will be either not initiated or aborted.
- If the ego vehicle is on the lane marker, the next lateral action will choose from *change lane to right* and *change lane to the left*, which means the ego vehicle would not stay on the lane marker.

2.3.6 Training Setups

The experiments are conducted with hyperparameters listed in Table 2.1. All three exploration methods are learning policies with the same deep network structure.

Table 2.1 Implementation Hyperparameters

	Description	Value
z	The dimension of the CVAE latent variable z	6
σ^2	Hyperparameter for CVAE decoder	$1/2\pi$
η_0	Weight factor in Equation (2.19) for r^{sp}	1.0
E	Number of evaluation episodes	10
γ	Discount factor	0.9
Δt	Sampling time	0.1 sec
ρ	Learning rate	1×10^{-6}
ε_0	Starting value for ε-greedy exploration	0.2
C	Annealing factor for ε-greedy exploration	2×10^{-6}

2.4 Training Results

In this section, we compare our proposed CVAE model-based exploration strategy with two baseline exploration methods described in Section 2.3.5 .

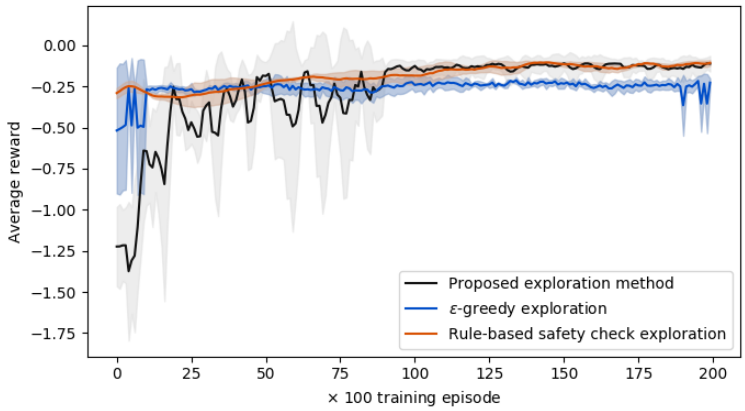


Figure 2.8 Average reward from evaluation roll-outs confidence bound

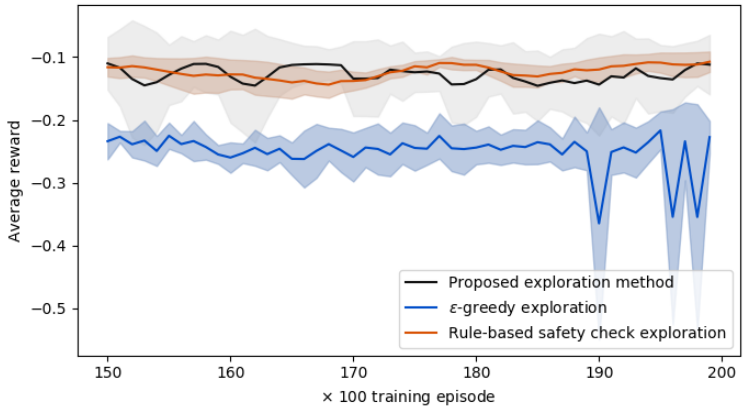


Figure 2.9 Average reward of last 50 episodes with the confidence bound

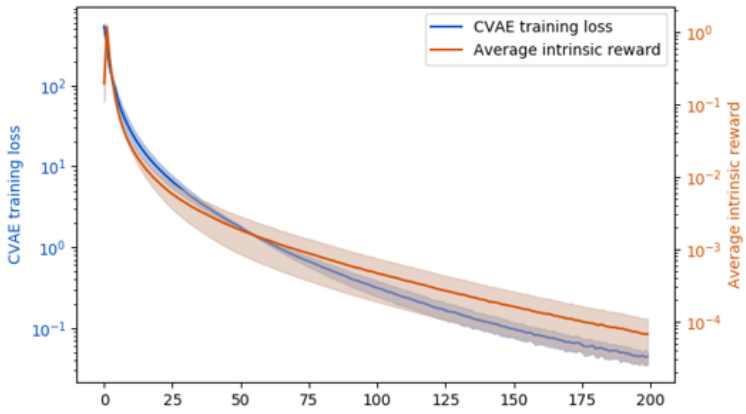


Figure 2.10 Average intrinsic reward and the CVAE training loss over training iterations

During training, the updated policy is tested by random rollouts using the environment reward r^e only. Figure 2.8 reports the average reward per action for ten evaluation rollouts. The training curve obtained by rule-based safety check exploration has very narrow confidence intervals and converges fast. The rule-based safety check helps the agent collecting smart data, and the reinforcement learning part is more or less like a fitting process. Meanwhile, the training curve of ε-greedy shows that by only randomly choosing an action, the agent is learning slowly. While for our proposed CVAE model-based exploration method, the result shows that its performance is as good as our target rule-based safety check exploration method, and during the early stage, it is trying to explore unfamiliar states, which leads to sudden drops in the average reward curve. Also, in Figure 2.9, the average reward of the last 50 episodes are reported. Our method performs twice as well as the ε-greedy baseline. In the end, the ε-greedy method converges to a suboptimal policy that learns to stay in one lane without any intention of a lane change. And if the agent learns to perform good lane following, the average reward would be around -0.2.

The intrinsic reward from CVAE and the CVAE training loss is also reported in Figure 2.10. During training, the CVAE becomes more and more accurate while the intrinsic reward vanishes at the end of the training. Therefore, the agent is eventually learning a good policy corresponding to the environment reward.

These experiments confirm that the work presented in this chapter outperforms the baseline ε-greedy method and converge almost as fast as the oracle method exploration. As shown in Figure 2.11, the AV agent trained by the baseline ε-greedy method will converge to a no-lane change policy. Even if cars in the adjacent lane travel faster, the AV agent will not change lanes to achieve a higher speed. This is due to the poor exploration during learning, which leads the AV agent to a local optimal no-lane change policy. While for the AV agent trained by the model-based exploration method, as shown in Figure 2.12, it can change lane to the adjacent lane where cars travel faster. These results show that the model-based exploration method can find the global-optimal policy (i.e., the oracle policy trained by rule-based safety check exploration), while when using the baseline ε-greedy method, the AV agent will easily be stuck in a local-optimal policy.

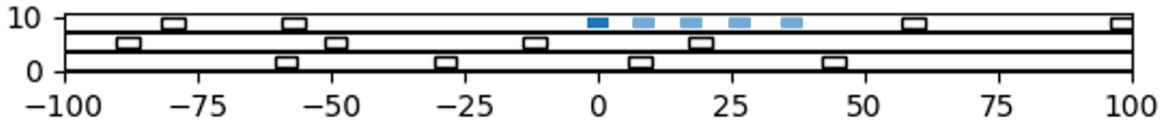


Figure 2.11 Animation result of ε-greedy method

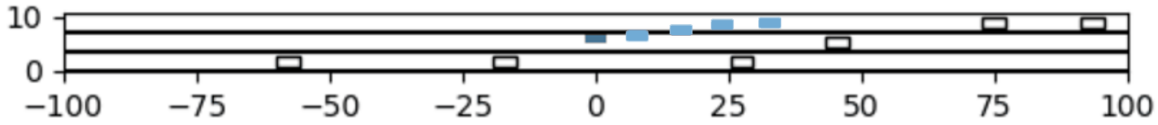


Figure 2.12 Animation result of model-based exploration method

2.5 Summary

In this work, we proposed a type of model-based exploration method via intrinsic reward. In particular, an environment transition model structured as a CVAE model with vehicle kinematics is learned during network training. In parallel, this model is encoded in a surprise-based intrinsic reward exploration for the policy training. The experiments we conduct show that the model-based exploration method we proposed leads to a faster convergence solution (i.e., better data efficiency) than the baseline ε-greedy approach and as good as the oracle rule-based safety check exploration method.

As shown in the simulation results, the policy learned by the baseline ε-greedy method will converge to a local-optimal policy that the AV agent will stay in one lane even when the cars in the adjacent lane travel faster. The model-based exploration method we developed can help the agent explore the state that runs contrary to their understanding of the environment and thus result in a thorough exploration. And as shown in the simulation result, the AV agent learns to change lane to travel both safer and faster.

The method we developed shows both theoretical and practical advantages in solving the discretionary lane change problem. The AEB used in short-horizon safety check will be evaluated in Chapter 3, and the trained policy will be evaluated in Chapter 4.

Chapter 3 Evaluation of the Autonomous Vehicle's Policy Using Subset Simulation

In this chapter, we evaluate the Advanced Emergency Braking (AEB) system implemented as the lower level safeguard controller in the short horizon safety check in Section 2.3.5. The evaluation problem under the first setup of assumptions (described in Section 1.4) is studied. Given the naturalistic driving data from the SPMD database, environment stochastic models were built. The approach developed in this chapter is based on the Subset Simulation (SS) method. The SS approach is demonstrated in the lane change scenario with two environment stochastic models. The crash rate and accelerated rate are calculated and compared to the baseline Importance Sampling (IS) method.

3.1 Literature Reviews on Evaluation Approaches and Their Limitations

As briefly mentioned in Section 1.2.2, variation reduction techniques have been proposed as a solution to the limitations of N-FOTs and other approaches [5]. Here we expound on variation reduction techniques used in past studies and their limitations.

In general, Monte Carlo simulations were very inefficient since much real-world driving consists of non-safety-critical interactions between the host vehicle and its surrounding vehicles. To address this limitation, Zhao et al. [5], [69], [70] applied the IS method to estimate the probability of rare events and use that to achieve significantly faster AV evaluations than the conventional Monte Carlo approach. Instead of relying upon the initial distribution of naturalistic driving data, the IS estimator samples according to the Importance Sampling Distribution (ISD). This distribution is the result of a bijective transformation, and therefore testing under this framework can focus on safety-critical scenarios while retaining the probability of such scenarios. Consequently, results under IS can be interpreted in the context of the original distribution, and thus the probability of rare events can be accurately estimated much more quickly. This technique has been used for testing AEB in lane changing scenarios in [70] and was found to accelerate testing by around 50 times compared with the Crude Monte Carlo method.

3.1.1 Importance Sampling Method

To introduce the importance sampling method, let us first formulate the problem under the language of mathematics. The objective of using the IS method is to accurately and efficiently approximate rare events probability (failure probability P_F in our application)

$$P_F = \mathbb{P}(\varepsilon), \qquad \varepsilon \subset \Omega \tag{3.1}$$

where Ω denotes the variable space of all possible events, and ε denotes the set of rare events. The indicator function of the event ε is defined as

$$I_\varepsilon(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{x} \in \varepsilon \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

where $\boldsymbol{x}$ denotes the vector of random variables that describes the surrounding vehicles. Let $f(\boldsymbol{x})$ denote the joint Probability Density Function (PDF) of the environment distribution, then the failure probability P_F can be written as

$$P_F = \mathbb{E}_f[I_\varepsilon(\boldsymbol{x})] = \int_{\boldsymbol{x}\in\varepsilon} I_\varepsilon(\boldsymbol{x}) f(\boldsymbol{x}) \mathrm{d}\mathbf{x} \tag{3.3}$$

where $\mathbb{E}_f[I_\varepsilon(\boldsymbol{x})]$ means the expectation of $I_\varepsilon(\boldsymbol{x})$ and $\boldsymbol{x}\sim f(\boldsymbol{x})$. P_F can be estimated by the **Crude Monte Carlo** (CMC) simulation as

$$\hat{P}_F^{CMC} = \frac{1}{N}\sum_{i=1}^{N} I_\varepsilon(x_i), \quad x_i \sim f(\boldsymbol{x}) \tag{3.4}$$

where N is the total number of samples used to estimate the P_F, and x_i is the i^{th} independent and identically distributed (i.i.d.) sample from the environment distribution $f(\boldsymbol{x})$. The CMC estimation $\hat{P}_F^{CMC}$ is an unbiased estimator [101] of P_F with mean $\mathbb{E}\left[\hat{P}_F^{CMC}\right]$ and variance $\mathbb{V}\left[\hat{P}_F^{CMC}\right]$ easily being derived as

$$\mathbb{E}\left[\hat{P}_F^{CMC}\right] = P_F, \text{ and } \mathbb{V}\left[\hat{P}_F^{CMC}\right] = \frac{P_F(1-P_F)}{N}. \tag{3.5}$$

In reliability analysis, the standard measure of the accuracy of an unbiased estimator $\hat{P}_F$ is its *coefficient of variation (c.o.v.)* [101], defined as $\delta\left(\hat{P}_F\right) = \sqrt{\mathbb{V}\left[\hat{P}_F\right]}/\mathbb{E}\left[\hat{P}_F\right]$. Therefore, we can calculate the c.o.v. expression for CMC estimator:

$$\delta\left(\hat{P}_F^{CMC}\right) = \sqrt{\mathbb{V}\left[\hat{P}_F^{CMC}\right]}/\mathbb{E}\left[\hat{P}_F^{CMC}\right] = \sqrt{\frac{1-P_F}{NP_F}}. \tag{3.6}$$

The idea of c.o.v. will permeate this chapter as a standard measure. It is obvious that the c.o.v. of the CMC estimator is related to the failure probability P_F and the total number of samples N. Considering estimating the probability of rare events, i.e. $P_F \ll 1$, then $\delta\left(\hat{P}_F^{CMC}\right) \approx 1/\sqrt{NP_F}$. For example, if $P_F = 10^{-7}$ (which is the magnitude of human driver crash rate), and if the c.o.v. $\delta\left(\hat{P}_F^{CMC}\right) = 10\%$ is desirable, then $N = 10^9$ samples are required. That is the reason we introduce the Importance Sampling (IS) technique.

The **IS technique** [102] aims to increase the estimation accuracy by constructing some Importance Sampling Distribution (ISD). The basic idea of the IS method is to take advantage of the information about the rare event to generate samples that lie more frequently in the important region, or in our application, the "danger region." We denote this ISD as $f^*(\boldsymbol{x})$, and we can redo the analysis as in CMC. The failure probability P_F can be derived as

$$P_F = \int_{\boldsymbol{x}\in\varepsilon} I_\varepsilon(\boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x} = \int_{\boldsymbol{x}\in\varepsilon} \frac{I_\varepsilon(\boldsymbol{x}) f(\boldsymbol{x})}{f^*(\boldsymbol{x})} f^*(\boldsymbol{x}) d\boldsymbol{x} = \mathbb{E}_{f^*}\left[\frac{I_\varepsilon(\boldsymbol{x}) f(\boldsymbol{x})}{f^*(\boldsymbol{x})}\right]. \tag{3.7}$$

Now the IS estimator can be constructed similarly as

$$\hat{P}_F^{IS} = \frac{1}{N}\sum_{i=1}^{N} \frac{I_\varepsilon(x_i) f(x_i)}{f^*(x_i)} = \frac{1}{N}\sum_{i=1}^{N} I_\varepsilon(x_i)\, L(x_i), \quad x_i \sim f^*(\boldsymbol{x}) \tag{3.8}$$

where x_i is the i^{th} i.i.d. sample from the ISD $f^*(\boldsymbol{x})$, and $L(x_i) = f(x_i)/f^*(x_i)$ is the importance weight of the sample x_i. The IS estimator converge to P_F as $N \to \infty$ by the strong law of large numbers, if and only if the support of $f^*(\boldsymbol{x})$, i.e., the domain where $f^*(\boldsymbol{x}) > 0$, contain the support of $I_\varepsilon(x_i) f(x_i)$. By choosing the ISD appropriately, the IS method can obtain an estimator with a smaller variance. The variance of the IS method can be derived as

$$\begin{aligned} \mathbb{V}\left[\hat{P}_F^{IS}\right] &= \frac{1}{N^2}\sum_{i=1}^{N} \mathbb{V}_{f^*}\left[I_\varepsilon(\boldsymbol{x}) L(\boldsymbol{x})\right] \\ &= \frac{1}{N}\left(\mathbb{E}_{f^*}\left[I_\varepsilon(\boldsymbol{x}) L^2(\boldsymbol{x})\right] - P_F^2\right) \end{aligned}. \tag{3.9}$$

It is straightforward to derive that the optimal choice of ISD which minimize the variance of IS estimator in Equation (3.6)

$$f_{opt}^{*} = \frac{I_{\varepsilon}(\boldsymbol{x}) f(\boldsymbol{x})}{P_F} = f(\boldsymbol{x} | x \in \varepsilon) \tag{3.10}$$

which is simply the original PDF $f(\boldsymbol{x})$ conditional on the rare event domain. The f_{opt}^{*} is called the *zero-variance distribution*, which zeroes the variance of the IS estimator. Therefore, with zero-variance distribution as the ISD, the IS estimator can accurately approximate P_F at 100% confidence level with only 1 sample. This is the big promise of the IS method.

3.1.2 Limitations and Motivations

However, *zero-variance distribution* is just a theoretical result. Finding this distribution for the IS estimator is not easy. It requires knowledge of the AV's algorithm, which is not available under our assumptions, nor is it a practical requirement in general. If we treat the AV as a black-box system, some Adaptive Importance Sampling (AIS) technique is needed. The Cross-Entropy (CE) method [103] is a normal choice of AIS techniques, which use the Monte Carlo method to minimize the CE between the optimal ISD (i.e., zero-variance distribution) and the proposed ISD

$$H\left(f_{opt}^{*}(\boldsymbol{x}), f^{*}(\boldsymbol{x})\right) = -\int_{\Omega} f_{opt}^{*}(\boldsymbol{x}) \log\left(f^{*}(\boldsymbol{x})\right) d\boldsymbol{x} \tag{3.11}$$

where $H\left(f_{opt}^{*}(\boldsymbol{x}), f^{*}(\boldsymbol{x})\right)$ denotes the CE. And substituting Equation (3.10) into Equation (3.11), we have

$$H\left(f_{opt}^{*}(\boldsymbol{x}), f^{*}(\boldsymbol{x})\right) = -\int_{\Omega} \frac{I_{\varepsilon}(\boldsymbol{x}) f(\boldsymbol{x})}{P_F} \log\left(f^{*}(\boldsymbol{x})\right) d\boldsymbol{x}\,. \tag{3.12}$$

Therefore, minimizing the CE is to maximize $\int_{\Omega} \frac{I_{\varepsilon}(\boldsymbol{x}) f(\boldsymbol{x})}{P_F} \log\left(f^{*}(\boldsymbol{x})\right) d\boldsymbol{x}$ which is proportional to $\int_{\Omega} I_{\varepsilon}(\boldsymbol{x}) f(\boldsymbol{x}) \log\left(f^{*}(\boldsymbol{x})\right) d\boldsymbol{x}$. Furthermore, we assume that the $f^{*}(\boldsymbol{x})$ can be determined by parameter θ, and thus the objective function of the CE method can be rewritten as

$$\max_{\theta} \int I_{\varepsilon}(\boldsymbol{x}) f(\boldsymbol{x}) \log\left(f^{*}(\boldsymbol{x}; \theta)\right) d\boldsymbol{x}\,. \tag{3.13}$$

This can be solved recursively by MC sampling method, and as derived in [104], the k^{th} step update equation for θ is

$$\theta_k = \arg\max_{\theta} \frac{1}{N} \sum_{i=1}^{N} I_{\varepsilon}(x_i) \frac{f(x_i)}{f^*(x_i;\theta_{k-1})} \log f^*(x_i;\theta), \quad x_i \sim f^*(\boldsymbol{x};\theta_{k-1}) \tag{3.14}$$

where the x_i is the i^{th} i.i.d. sample from the $k-1$ step ISD $f^*(\boldsymbol{x};\theta_{k-1})$ and N is the total number of samples for k^{th} step. However, it requires a lot of additional samples before the testing procedure, which will drain the accelerated rate. This is the first motivation for the research in this chapter.

Furthermore, the IS method suffers from the high dimension degeneration problem [105], or the *curse of dimension* problem. A geometric explanation as to why IS is often inefficient in high dimensions is given in [106], and the theoretical analysis of this problem is addressed in Section 2.6.6.1 in [73]. Since the illustration of the problem will take too much effort, here we just give the conclusion and simple explanations. The conclusion is that the failure probability P_F is found to be severely underestimated in high dimension environment from their numerical experiments [106], the underestimation becoming worse as the dimension increases. A simple explanation is that even if the ISD $f^*(\boldsymbol{x})$ is close to the optimal ISD $f^*_{opt}(\boldsymbol{x})$, it still has some possibility that the support of $f^*(\boldsymbol{x})$ does not cover the support of $f^*_{opt}(\boldsymbol{x})$ in some dimension, which will lead to underestimation of the P_F. And the higher the problem dimension, the worse IS underestimates.

To solve the curse of dimensionality problem, **Sequential Importance Sampling** (SIS) techniques can be used. In the SIS methods [107], the ISD is iteratively refined to represent rare events. However, fitting ISD requires predefining the model or structure of this ISD. Knowing nothing about the "danger region" of the testing AV leaves no clue for the structure of this ISD. For example, if we choose the Gaussian Mixture Model (GMM) as the structure for this ISD, what should be the component number of this GMM? How can we be sure this GMM is the "best model" and will not overfit? We cannot ensure this GMM is the best model without further Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC) testing. Moreover, we do not need the posterior distribution. We only need the failure probability P_F. This is another motivation for the following research.

3.2 Variance Reduction Techniques: Subset Simulation

The above-mentioned limitations of the IS method lead us to study another sampling method, the Subset Simulation (SS), to address the black-box and high-dimensionality problems. SS is an advanced stochastic simulation method for estimating the low failure probability of a system based on the Markov Chain Monte Carlo method [73], [108], [109]. It has been used to estimate the structural reliability of civil, aerospace, and nuclear systems. Moreover, SS has proven useful in other applications such as sensitivity analysis, design optimization, and uncertainty quantification [73].

3.2.1 Subset Simulation Algorithm

The basic idea behind the Subset Simulation (SS) method is to represent a very small probability (the failure probability in our application) P_F as a product of relative larger probabilities of "more-frequent" events. Then the larger probabilities are estimated separately. To construct a sequence of sets of events, let us consider a potentially non-explicit function $Y(x)$ to indicate the performance of the system. The rare event set ε in Equation (3.1) can be rewritten as

$$\varepsilon = \{\boldsymbol{x}: Y(\boldsymbol{x}) < b\} \tag{3.15}$$

where b is the threshold of the performance for the rare event set. Then we can construct a sequence of sets of events as

$$\Omega \equiv \varepsilon_0 \supset \varepsilon_1 \supset \varepsilon_2 \ldots \supset \varepsilon_M \equiv \varepsilon \tag{3.16}$$

where M is the total number of *levels* in the simulation and $\varepsilon_m = \{\boldsymbol{x}: Y(\boldsymbol{x}) < b_m\}$ is the subset at the level m simulation. Then we have $b = b_M < b_{M-1} < \cdots < b_1$ and $\varepsilon = \bigcap_{m=0}^{M} \varepsilon_m$. Therefore, as far as we can construct such a sequence of nested subsets, the failure probability P_F can be calculated as a product of conditional probabilities

$$P_F = \mathbb{P}(\varepsilon) = \mathbb{P}\left(\bigcap_{m=0}^{M} \varepsilon_m\right) = \prod_{m=1}^{M} \mathbb{P}(\varepsilon_m|\varepsilon_{m-1}) = \prod_{m=1}^{M} P_m \tag{3.17}$$

where $P_m \equiv \mathbb{P}(\varepsilon_m|\varepsilon_{m-1})$ is the conditional probability at the m^{th} level and $\mathbb{P}(\varepsilon_1|\varepsilon_0) = \mathbb{P}(\varepsilon_1)$. In this way, the original problem of estimating a small failure probability P_F becomes M

intermediate problems corresponding to larger conditional probabilities. In the implementation of SS, the total number of subsets (or levels) M and the values of intermediate thresholds $\{b_m\}$ are chosen adaptively.

The next question is to estimate these conditional probabilities. First, we can further define the indicator function for each level

$$I_{\varepsilon_m}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{x} \in \varepsilon_m, \text{ i.e., } Y(\boldsymbol{x}) < b_m \\ 0, & \text{otherwise} \end{cases} \tag{3.18}$$

and the PDF of the conditional distribution of each level can be derived as

$$p_m(\boldsymbol{x}) \equiv f(\boldsymbol{x}|\varepsilon_{m-1}) = \frac{I_{\varepsilon-1}(\boldsymbol{x}) f(\boldsymbol{x})}{\mathbb{P}(\varepsilon_{m-1})} \tag{3.19}$$

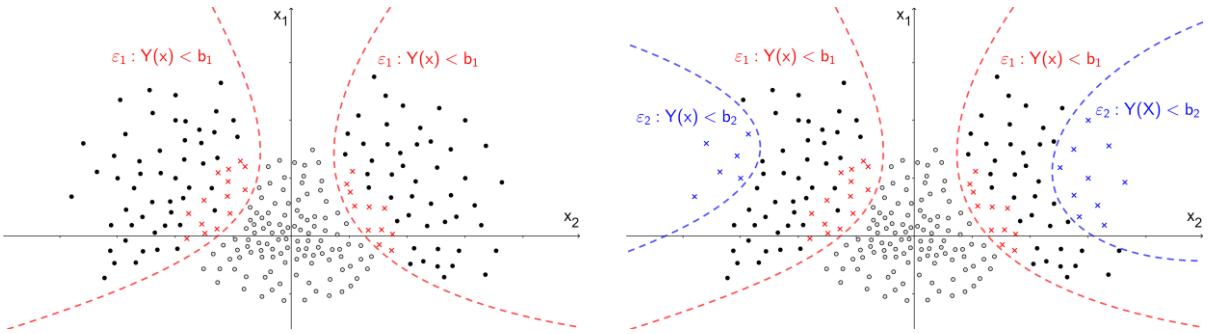
Since $\mathbb{P}(\varepsilon_0) = 1$, the first probability can be estimated by the CMC method. Estimating the remaining probability is more challenging and need to sample from the conditional distribution in Equation (3.19). This seems to be a trivial task, noticing that a sample from $p_m(\boldsymbol{x})$ is just one drawn from $f(\boldsymbol{x})$ that lies in ε_{m-1}. However, it is not efficient to draw a sample from $f(\boldsymbol{x})$ and abandon it if it is not lies in ε_{m-1}. Instead, in standard SS, samples from the conditional distribution $p_m(\boldsymbol{x})$ are generated by the Modified Metropolis Algorithm (MMA) [108], which belongs to the family of MCMC. Details of MMA will be elaborated in the next Section 3.2.2 .

The splitting strategy for each simulation level (m) is introduced here first and summarized as follows:

1. The estimation of first level ($m = 1$) conditional probability is conducted by CMC, which directly draw N samples $x_1^{(1)}, \ldots, x_N^{(1)}$ from the original environment distribution $f(\boldsymbol{x})$.
2. The performance function $Y(\boldsymbol{x})$ is used to characterize failures; $Y(x_i^{(1)})$ is evaluated for each $x_i^{(1)}$, which is the i^{th} sample from the CMC sampling; all $Y(x_i^{(1)})$ are then sorted in ascending order to get the list $\{y_1^{(1)} \leq \ldots \leq y_N^{(1)}\}$.
3. Set the P_1 (defined in Equation (3.17)) percentile of the list $\{y_1^{(1)} \leq \ldots \leq y_N^{(1)}\}$, denoted as b_1, to be the threshold for the 2nd subset level. This means that $\varepsilon_1 = \{\boldsymbol{x}: Y(\boldsymbol{x}) < b_1\}$; those samples inside the ε_1 are "seeds" for the 2nd level. We denote seeds using $\{\theta_j^{(1)}\}$.

4. In the 2nd level, we sample from the conditional distribution $p_1(\boldsymbol{x})$. It is inefficient to use CMC; thus, we choose the MMA sampling method for this task.
5. After collecting samples $x_1^{(2)},\ldots,x_N^{(2)}$, repeat step 2 and obtain b_2 and $\varepsilon_2 = \{\boldsymbol{x}: Y(\boldsymbol{x}) < b_2\}$. Once again, those samples inside ε_2 are seeds for the 3rd level and denoted as $\{\theta_j^{(2)}\}$.
6. Repeat step 3 to 5 for samples of the next level until $b_m \leq b$ or $m+1 > M$.

This process is illustrated in a two-dimension variable space in Figure 3.1. In Figure 3.1 (a), the hollow dots (circles) are the samples from the original distribution $f(\boldsymbol{x})$. They are then tested by the performance function $Y(\boldsymbol{x})$ and the testing results are sorted in ascending order. The red crosses (are also sampled from the $f(\boldsymbol{x})$) in Figure 3.1 (a) are the top P_1 percent results and thus become the seeds for level 1. In the subset ε_1, level 1 samples (the black dots in Figure 3.1 (a)) are drawn from the seeds by MMA. Then level 1 samples are also tested by performance function $Y(\boldsymbol{x})$ and the testing level 1 results are also sorted in ascending order. The blue crosses in Figure 3.1 (b) are the top P_2 percent of the level 1 results, and thus become the seeds for level 2. Repeat these steps until the terminal condition reached.



(a) Level 0 – Level 1 (b) Level 0 – Level 2

Figure 3.1 Subset Simulation Process

3.2.2 Modified Metropolis Algorithm

In the procedure outlined in Section 3.2.1 , the critical problem is the efficient sampling at each level from the conditional probability distribution $p_m(\boldsymbol{x}) \equiv f(\boldsymbol{x}|\varepsilon_{m-1})$, i.e. to estimate

$$P_m = \int_{x\in\varepsilon} p_m(\boldsymbol{x})d\boldsymbol{x} = \int_{x\in\varepsilon} \frac{I_{\varepsilon-1}(\boldsymbol{x})f(\boldsymbol{x})}{\mathbb{P}(\varepsilon_{m-1})}d\boldsymbol{x}\,. \tag{3.20}$$

To approach this issue, we consider Markov Chain Monte Carlo (MCMC), which is a class of sampling methods for distributions that cannot be directly sampled efficiently. The basic idea of this method is to construct a Markov Chain whose stationary distribution is the one of interest. By drawing from the Markov Chain, the samples will, in the end, be distributed with the conditional probability distribution $p_m(\boldsymbol{x})$. The MCMC used in SS is the Modified Metropolis Algorithm (MMA).

The MMA algorithm is a component-wise version of the original Metropolis algorithm [110]. It is specifically tailored for sampling from high-dimensional conditional distributions. This approach uses the seeds as described before and evolves according to the *proposal distribution* $q_k(\cdot\,|\,\cdot)$, $k = 1, \ldots, K$, which corresponds to the k^{th} dimension of the original distribution $f(\boldsymbol{x})$. For level m, we have $N \times P_{m-1}$ seeds, where N is the number of samples from the previous level, and P_{m-1} is the level probability. The MMA process at each level is described in Algorithm 3.1 and illustrated in Figure 3.2 (in two-dimension variable space).

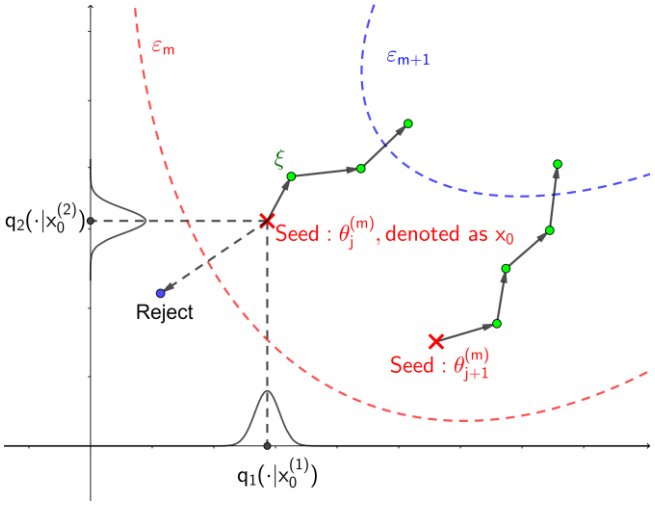


Figure 3.2 Schematic Diagram of Modified Metropolis Algorithm

Algorithm 3.1: Modified Metropolis Algorithm for each seed $\theta_j^{(m-1)}$ in level m

Input:

Initial state: $\theta_j^{(m-1)}$, which is denoted as x_0 for each Markov Chain.
Total number of states of the Markov Chain: N_c.
Original distribution: $f_k(\cdot)$ for dimension k.
Proposal distribution: $q_k(\cdot \mid \cdot)$ for dimension k.

for $i = 1, \dots, N_c$ **do**

Generate candidate state ξ

for $k = 1, \dots, K$ **do**

Sample $\hat{\xi}_k \sim q_k(\cdot \mid x_{i-1}^{(k)})$

Calculate the acceptance ratio for MCMC

$$r = \frac{f_k(\hat{\xi}_k) \cdot q_k(x_{i-1}^{(k)} \mid \hat{\xi}_k)}{f_k(x_{i-1}^{(k)}) \cdot q_k(\hat{\xi}_k \mid x_{i-1}^{(k)})} \tag{3.21}$$

Accept or reject $\hat{\xi}_k$:

$$\xi_k = \begin{cases} \hat{\xi}_k, & \text{with probability } \min(r, 1) \\ x_{i-1}^{(k)}, & \text{with probability } 1 - \min(r, 1) \end{cases} \tag{3.22}$$

Obtain $\xi = [\xi_1, \dots, \xi_K]^T$

Check whether $\xi \in \varepsilon_{m-1}$ by testing. Accept or reject ξ:

$$x_i = \begin{cases} \xi, & \text{if } \xi \in \varepsilon_{m-1} \\ x_{i-1}, & \text{otherwise} \end{cases} \tag{3.23}$$

Output: Samples $x_0, \dots, x_{N_c-1}$, N_c states of a Markov Chain for each seed $\theta_j^{(m-1)}$ in level m.

By applying the MMA algorithm, the resulting stationary distribution will be the conditional distribution $p_m(\boldsymbol{x})$. Here we observe that the total number of samples in level m is $N \times P_{m-1} \times N_c$. For convenience, if we set $N_c = P_{m-1}^{-1}$, then at each level, we will have the same number of total samples. The MCMC method is known to handle high-dimensional stochastic models efficiently, which is an important consideration because the stochastic model for a realistic environment of an AV can be complicated.

One important assumption of this MMA algorithm is that the K dimension random variables are independent. This assumption is not a limitation since, in simulation, one always starts from independent variables to generate correlated excitation histories. The modification of the original distribution $f(\boldsymbol{x})$ for MMA will be elaborated in Section 3.3.3 . Other parameter selection will be elaborated in Section 3.4.2 .

It is derived in [73], [108] that, given the failure probability P_F, P_m, and the total number of samples N, the c.o.v. of the SS estimator can be derived as

$$\delta^2\left(\hat{P}_F^{SS}\middle|P_F, P_m, N\right) = \frac{(1+\gamma)(1-P_m)}{NP_m(\ln P_m^{-1})^r}(\ln P_F^{-1})^r \tag{3.24}$$

where $2 \le r \le 3$ and γ is approximately a constant that depends on the state correlation of the Markov chain at each level. Numerical experiments show that $r = 2$ gives a good approximation to the c.o.v. and that $\gamma \approx 3$ if the variance of the proposal distribution is chosen appropriately [73], [108], [111]. This will be elaborated in Section 3.4.2 . As shown in Equation (3.6), the c.o.v. of CMC is $\delta^2\left(\hat{P}_F^{CMC}\right) \propto P_F^{-1}$, while for SS, $\delta^2(\hat{P}_F^{SS}) \propto (\ln P_F^{-1})^r$. This is the reason why SS can dramatically improve the efficiency of CMC in rare event simulation.

3.3 Naturalistic Driving Data and Environment Model

3.3.1 Naturalistic Driving Data

The lane change scenario is used as a case study to show the evaluation results and the advantages of SS over CMC and IS techniques. In the lane change scenarios, two vehicles are involved, which are the following vehicle and the lane change vehicle (as shown in Figure 3.3). In the US, around 4 ~ 10% of all freeway crashes are related to lane change [112], i.e., around 240,000 to 610,000 reported lane-change crashes every year [113]. Therefore, we aim to develop an advanced evaluation procedure for AV systems under lane change scenarios.

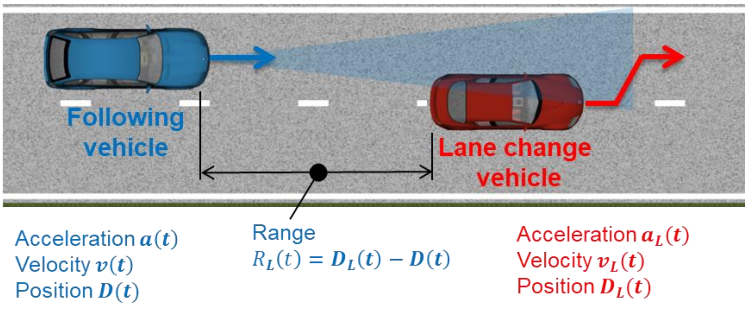


Figure 3.3 Lane Change Scenario Features

To test the AV system under the lane-change scenario, we first need to analyze the human drivers' behaviors. Early studies focus mainly on the gap acceptance [114] and other driver models [115] with short horizons and limited control settings. As more and more naturalistic driving data were collected, researchers started to analyze the lane change behavior in depth. Zhao et al. [116] analyzed the safety-critical features in mandatory and discretionary lane changes for heavy trucks. Fitch et al. [117] studied 100 cars to analyze the lane change events and crashes. In [5], Zhao built a lane change statistical model from the Safety Pilot Model Deployment (SPMD) project and demonstrated its usefulness for evaluating the safety of an Advanced Emergency Braking (AEB) system. The model contains only three variables, which are the lane change vehicle's velocity at a lane change $v_L(t_{LC})$, the range at lane change $R_L(t_{Lc})$ and the Time-To-Collision (TTC) at lane change time which is defined as

$$TTC_L = -\frac{R_L}{\dot{R}_L} \tag{3.25}$$

where $\dot{R}_L$ is the derivative of R_L. Moreover, these three parameters are modeled as independent random variables and are generated for the testing separately. During the lane change, Zhao further assumed that the lane change vehicle's velocity would not change. This is also not very realistic. Therefore, in this work, we build another stochastic model that can capture lane change behaviors more comprehensively.

First, we queried the lane change scenarios from the SPMD database [61], [62]. As shown in Figure 3.4, instead of just focusing on the information at the lane change initiation, we record the entire lane change trajectory of the two cars. Following the process, we queried more than 400,000 lane change cases. One lane change case can be represented by three sets of time series data, which are the longitudinal velocity curve of the lane change vehicle and the following vehicle and the lateral position curve of the lane change vehicle, together with the initial range. To further simplify the model, we fit the two velocity curves with a 2nd order polynomial (examples are shown in Figure 3.5). And the lateral position curve is fitted by a half-sine function [118], which can be further reduced to only one parameter, i.e., the lane change duration. Together with the initial range, the lane change cases can be represented by eight variables (as listed in Table 3.1).

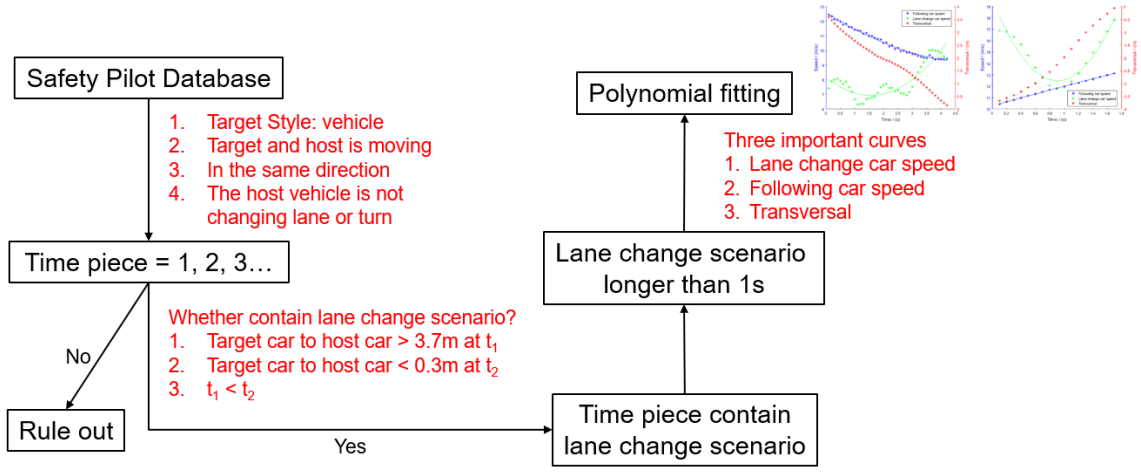
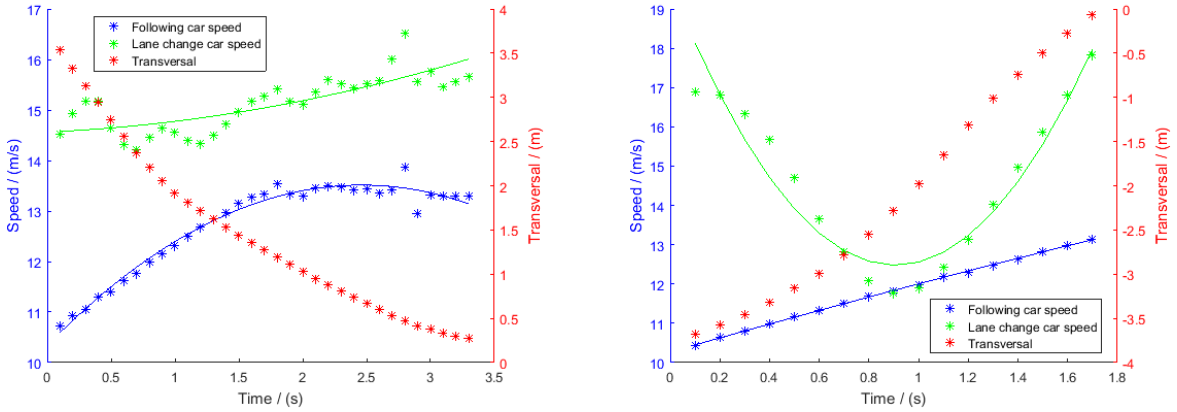


Figure 3.4 Data Processing for Querying the Lane Change Data



(a) Lane change from right to left

(b) Lane change from left to right

Figure 3.5 2nd Order Polynomial Fitting Examples

Table 3.1 Lane Change Model Features

$x_1 \sim x_3$	$x_4 \sim x_6$	x_7	x_8
Following vehicle's speed	Lane change vehicle's speed	Lane change duration	Initial range
Parameters of 2nd order polynomial: $\left(v_F^{(2)}\right)^2 t^2 + v_F^{(1)} t + v_F^{(0)}$	Parameters of 2nd order polynomial: $\left(v_L^{(2)}\right)^2 t^2 + v_L^{(1)} t + v_L^{(0)}$	T	R_0

Table 3.2 Correlation Matrix of the 8 Variables

	$v_F^{(0)}$	$v_F^{(1)}$	$v_F^{(2)}$	$v_L^{(0)}$	$v_L^{(1)}$	$v_L^{(2)}$	Duration T	Initial range R_0
$v_F^{(0)}$	1	-0.81	10^{-5}	0.20	-0.13	0.01	-0.02	-0.02
$v_F^{(1)}$	-	1	0.21	-0.26	0.27	0.28	0.18	0.26
$v_F^{(2)}$	-	-	1	-0.07	0.13	0.94	0.71	0.87
$v_L^{(0)}$	-	-	-	1	-0.88	-0.06	-0.05	-0.09
$v_L^{(1)}$	-	-	-	-	1	0.09	0.09	0.14
$v_L^{(2)}$	-	-	-	-	-	1	0.68	0.88
T	-	-	-	-	-	-	1	0.64
R_0	-	-	-	-	-	-	-	1

To select the right stochastic model for these eight variables, we first calculate the correlations between every pair of random variables. The correlation coefficient between two random variables X and Y is defined as

$$corr(X,Y) = \frac{\mathbb{E}[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y} \tag{3.26}$$

where μ_X and μ_Y are the expectations of random variables X and Y, while σ_X and σ_Y are the standard deviations of X and Y. And the correlation coefficient is symmetric, i.e., $corr(X,Y) = corr(Y,X)$. The value of a correlation coefficient ranges between -1 and $+1$. The correlation coefficient is $+1$ in the case of a perfect direct (increasing) linear relationship (correlation), -1 in the case of a perfect decreasing (inverse) linear relationship (anticorrelation) [119]. The correlation coefficient of each pair of the abovementioned eight variables is listed in Table 3.2

In Table 3.2. The eight variables are correlated, which means they are not independent. Some of these pairs have strong correlations (in red) or anticorrelations (in blue). Therefore, rather than using the model developed in [5], which has three independent variables, we choose the multivariate Gaussian Mixture Model (GMM), which can characterize the correlation.

3.3.2 Environment Model

In this work, we focused on capturing the entire lane change trajectory instead of just gap acceptance [77], [116], [120]. Recently, as more and more data became available, researchers start to use more complex models to represent human driver behavior. Angkititrakul et al. [121], [122] use a Gaussian Mixture Model (GMM) to characterize the driver behavior. Huang et al. [123] use GMM to develop a measure of robot driver etiquette in the car-following scenario.

Below we provide a simple introduction to the Gaussian Mixture Model (GMM) [124]. A GMM is a stochastic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions. For a given distribution, we use several Gaussian functions to fit it (as shown in Figure 3.6 (a)). The PDF for each component of the GMM is defined as

$$\phi_j(\boldsymbol{x}) = \frac{\pi_j}{\sqrt{2\pi \cdot \det(\Sigma_j)}} \exp\left(-\frac{1}{2}(x-\mu_j)^T {\Sigma_j}^{-1}(x-\mu_j)\right) \tag{3.27}$$

where μ_j is the mean value vector of the j^{th} component, Σ_j is the covariance matrix of the j^{th} component, and π_j is the weight for the j^{th} component.

Some data cannot be fitted by combinations of Gaussian distributions. Therefore researchers introduced another distribution called Generalized Gaussian Mixture Model (GGMM) [125], [126] and added a parameter λ to control the shape of different peak. The PDF of each component of GGMM is given by

$$T_j(\boldsymbol{x}) = \pi_j A(\lambda_j) \exp\left(-B(\lambda_j)\left|(x-\mu_j)^T {\Sigma_j}^{-1}(x-\mu_j)\right|^{\lambda_j/2}\right) \tag{3.28}$$

where the $A(\lambda_j)$ and $B(\lambda_j)$ are functions controlling the shape of each Gaussian distribution [126] and are defined as

$$A(\lambda_j) = \frac{\lambda_j \sqrt{\Gamma(3/\lambda_j)}}{2\det(\Sigma_j)\,\Gamma(1/\lambda_j)\sqrt{\Gamma(1/\lambda_j)}} \tag{3.29}$$

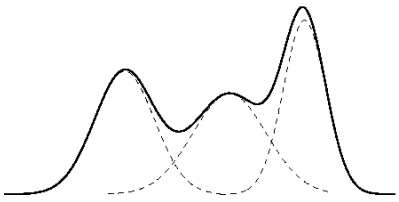
$$B(\lambda_j) = \left(\frac{\Gamma(3/\lambda_j)}{\Gamma(1/\lambda_j)}\right)^{\lambda_j/2} \tag{3.30}$$

In Equation (3.30), the $\Gamma(\cdot)$ denotes the gamma function. Each $\lambda_j \geq 0$ controls the shape of each GGMM component.

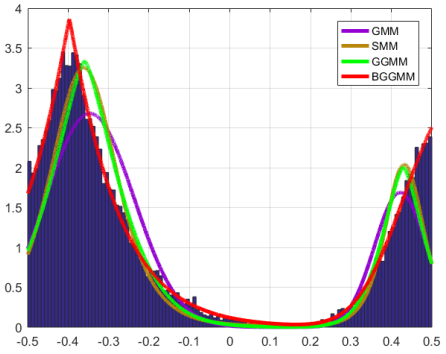
One drawback of GMM and GGMM is that their distributions are unbounded with a support region of $(-\infty, +\infty)$, while real data has boundaries. Therefore, Nguyen et al. [126] further introduced the Bounded Generalized Gaussian Mixture Model (BGGMM), which simply normalized the GGMM to the bounded support region while getting significant improvement. The PDF of each component of BGGMM is defined as

$$\Psi_j(\boldsymbol{x}) = \frac{T_j(\boldsymbol{x})H(\boldsymbol{x})}{\int T_j(\boldsymbol{x})} \tag{3.31}$$

where $H(\boldsymbol{x})$ is the bounded support region indicator which is equal to 1 within the bounded support region and 0 in other regions. The GMM, GGMM, and BGGMM fitting examples in [126] are reproduced and shown in Figure 3.6 (b), where the SMM is the Student's-t mixture model (SMM) that has similar performance as GGMM. The BGGMM fits the data the best.



(a) Schematic diagram of the GMM



(b) Examples for the fitting of GMM, SMM, GGMM and BGGMM [126]

Figure 3.6 Schematic Diagram for the GMM, GGMM, and BGGMM

We choose the Bounded Generalized Gaussian Mixture Model (BGGMM) as the stochastic model for the lane change scenarios since the eight variables, which represent the lane change scenario, have physical bounds. The BGGMM model is fitted by the standard Expectation Maximization (EM) algorithm [127]. However, we still need to select the component number for the BGGMM model. To quantify the model selection results, two information criteria are introduced, which are the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC). The AIC and BIC values are criteria for model selection

among a finite set of models. The model with the lowest AIC or BIC is preferred. They are based, in part, on the likelihood function [128]. The AIC value of a model is defined as

$$AIC = 2k - 2\ln\hat{L} \tag{3.32}$$

where k is the number of total parameters that need to be estimated, and $\hat{L}$ is maximum value of the likelihood function for the model. The AIC value rewards goodness of fit , but it also includes a penalty on model complexity. The BIC value is highly related to the AIC value and defined as

$$BIC = \ln(n) \cdot k - 2\ln\hat{L} \tag{3.33}$$

where n is the total number of data samples. For the same number of data samples, there is no big difference between AIC and BIC values.

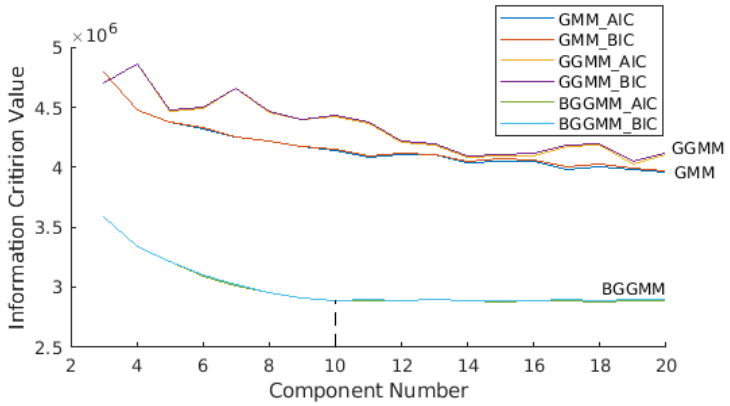
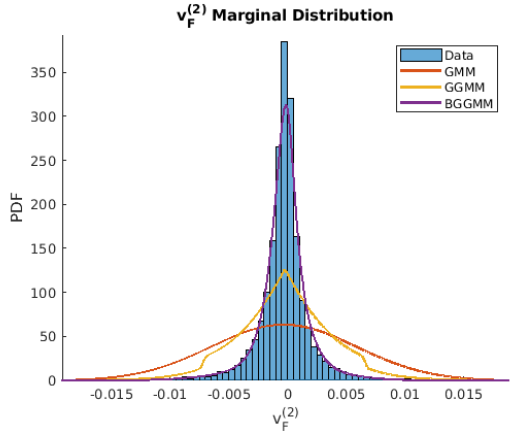
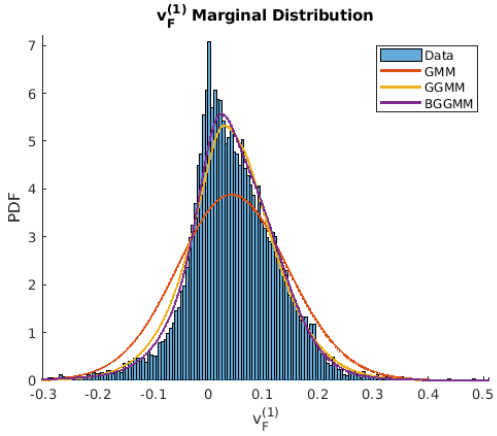


Figure 3.7 AIC and BIC Value of the BGGMM with Component Number from 3 to 20

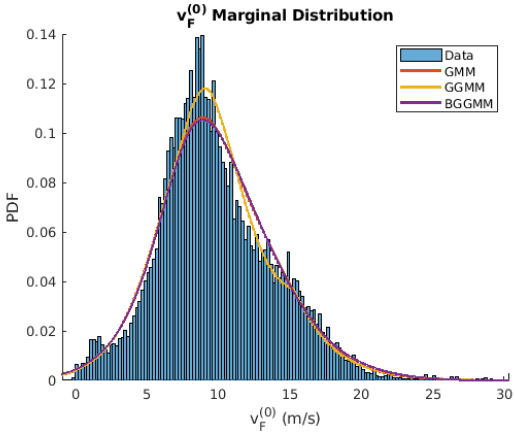
The AIC and BIC values are calculated for GMM, GGMM, and BGGMM with increasing component numbers from 3 to 20, and the results are shown in Figure 3.7. We choose the BGGMM with ten components as the stochastic lane change model. And the GMM, GGMM, and BGGMM with ten components fitting results are shown as the marginal distribution of each of the eight variables in Figure 3.8. Accurately capturing the distribution of the features of lane change will help estimate the crash rate of a specific system. In Section 3.4 , this BGGMM model will be used to sample test cases for the AEB system (details will be introduced in Section 3.4.1).



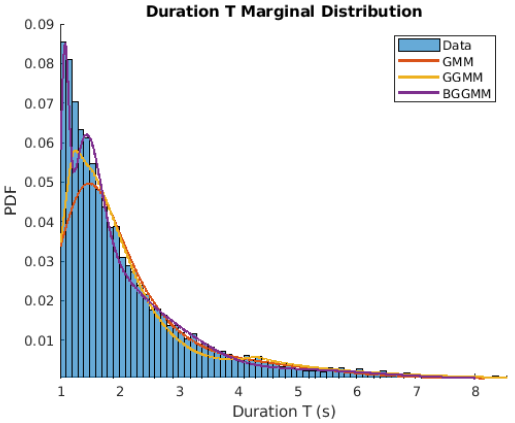
(a) Marginal distribution of $v_F^{(2)}$



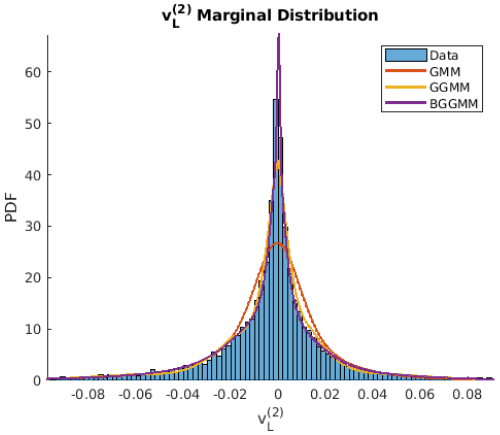
(b) Marginal distribution of $v_F^{(1)}$



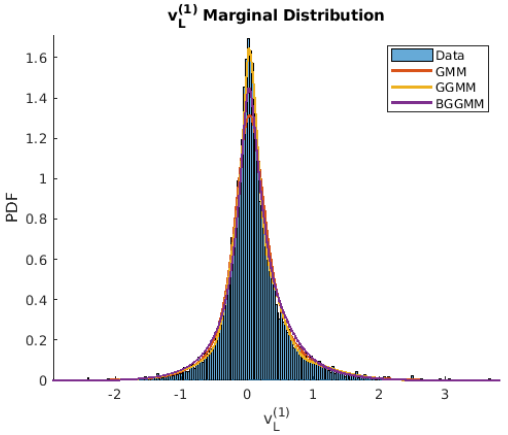
(c) Marginal distribution of $v_F^{(0)}$



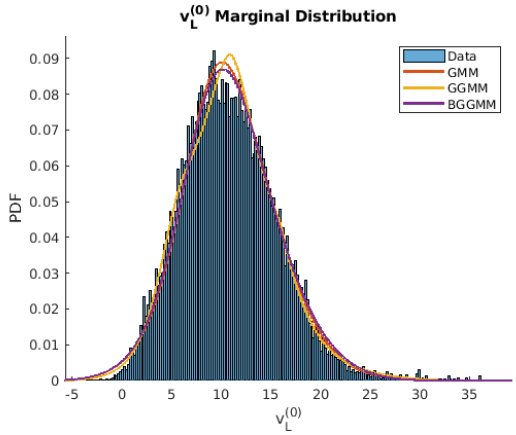
(d) Marginal distribution of duration T



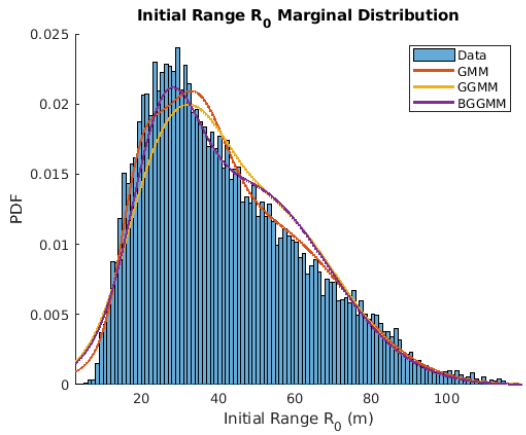
(e) Marginal distribution of $v_L^{(2)}$



(f) Marginal distribution of $v_L^{(1)}$



(g) Marginal distribution of $v_L^{(0)}$



(h) Marginal distribution of initial range R_0

Figure 3.8 BGGMM Fitting Results (Marginal Distribution PDF) for 8 Variables

3.3.3 Rosenblatt Transformation

The BGGMM is used to model the dependent lane change variables. As shown in Section 3.2.2 , one requirement for implementing MMA is that the distribution model has independent random variables. To address this issue, we introduce the Rosenblatt transformation method [129]. The Rosenblatt transformation is an isoprobabilistic transformation, which can transfer dependent random variables space X to a space U consisting of independent standard normal random variables by one-to-one mapping $T: X \to U$. The transformation is conduct as follows.

$$T_1: X \to Y = \begin{pmatrix} F_1(x_1) \\ \dots \dots \\ F_{k|1,\dots,k-1}(x_k|x_1,\dots,x_{k-1}) \\ \dots \dots \\ F_{K|1,\dots,K-1}(x_K|x_1,\dots,x_{K-1}) \end{pmatrix}$$

$$T_2: Y \to U = \begin{pmatrix} \Phi_1(y_1) \\ \dots \dots \\ \Phi_K(y_K) \end{pmatrix} \tag{3.34}$$

$$T = T_1 T_2: X \to U$$

where $F_{k|1,\dots,k-1}(\cdot)$ is the Cumulative Distribution Function (CDF) of the conditional random variable $x_k|x_1,\dots,x_{k-1}$, Φ_k is the CDF of the standard normal distribution, and the random variable vector $\boldsymbol{x}$ is in K-dimension. After the transformation, random variables in space U are independent and thus are ready for MMA. Moreover, the transformation is a one-to-one function,

and the inverse Rosenblatt transformation can be derived [130]. Therefore, even the MMA is conducted in the random variable space U, the results can be analysis in the original random variable space X.

3.4 Simulation and Results

3.4.1 Advanced Emergency Braking System under Test

To conduct the SS method for AV safety evaluation and compare it with the IS method, we test the ACC+AEB system previously studied in [5]. This model is also used as the lower level safeguard controller in the short horizon safety check in Chapter 2. In this evaluation, the lane change vehicle is controlled by a human driver (HV) who attempts to change lanes in front of the AV, while the following vehicle is the AV under test. The AV's AEB system is a black-box and unknown to the SS procedure and is taken from [5]. This AEB system is extracted from a production vehicle: 2011 Volvo V60.

As shown in Figure 3.9, the AV is controlled by the Adaptive Cruise Control (ACC) algorithm when the situation is normal and safe. The AEB algorithms become active when a threat is detected. If the AEB fails to prevent the crash, the simulation terminates. Otherwise, the control is returned to the ACC, and the test terminates as the lane change finishes. The ACC is approximated by a discrete Proportional-Integral controller, and the AEB is activated based on a threshold value TTC_{AEB} of "Time-To-Collision" defined in Equation (3.25). For the details of the ACC and AEB controller, please refer to [5].

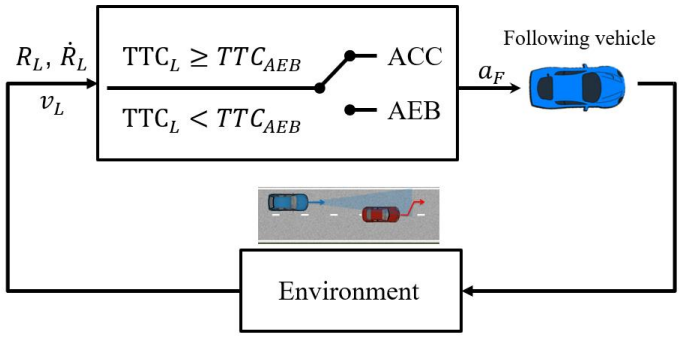


Figure 3.9 Layout of the AV Control System [5]

The ACC+AEB system is not a sophisticated AV system. It is selected as an example to demonstrate the difference between the SS evaluation method and the IS evaluation method.

3.4.2 Simulation Setups

In this section, we evaluate the AEB system with two different setups. In [5], Zhao evaluates the Volvo AEB system using a three-dimension independent random variable model. The variables are the lane change vehicle's velocity at the lane change initiation v_{L_0}, the range at the lane change initiation R_0 and the Time-To-Collision (TTC) at lane change time TTC_0. In the following, the SS is used to evaluate the AEB system using these three independent random variable model in order to compare with the results using the IS method [5]. The AEB system is also tested under the 8-variable BGGMM introduced in Section 3.3.2 . For easy differentiation, the baseline three independent random variables model is denoted as $f_{BL}(v_{L0}, TTC_0, R_0) = f_1(v_{L0})f_2(TTC_0)f_3(R_0)$ and the 8-variable BGGMM is denoted as $f_{BGGMM}(\boldsymbol{v_F}, \boldsymbol{v_L}, R_0, T)$, where $\boldsymbol{v_F} = [v_F^{(0)}, v_F^{(1)}, v_F^{(2)}]$ is the random variable vector for the parameters of the following vehicle's velocity, and $\boldsymbol{v_L} = [v_L^{(0)}, v_L^{(1)}, v_L^{(2)}]$ is the random variable vector for the parameters of lane change vehicle's velocity.

Before testing under this BGGMM, preprocessing is needed. When sampling for test cases, we can only sample the initial speed of the AV (i.e., the following vehicle) and have no control of its subsequent speed. Therefore, we need to calculate the marginal distribution of the AV's speed parameters other than the initial speed parameter ($v_F^{(0)}$). Thus, we sample from

$$f_{BGGMM}\left(v_F^{(0)}, \boldsymbol{v_L}, R_0, T\right) = \iint_{v_F^{(1)}, v_F^{(2)}} f_{BGGMM}(\boldsymbol{v_F}, \boldsymbol{v_L}, R_0, T) dv_F^{(1)}\, dv_F^{(2)} \tag{3.35}$$

Then we transform these dependent random variables to a space U consisting of independent standard normal random variables by the Rosenblatt transformation introduced in Section 3.3.3 . Using the Rosenblatt transformation, the SS can explore the space of U and inverse map to space X to test the AV. If variables of the $f_{BL}(v_{L0}, TTC_0, R_0)$ model are independent, then no Rosenblatt transformation is needed.

In the lane change scenario, the performance function $Y(\boldsymbol{x})$ is the minimum range during the entire lane change. Therefore, the set of actual crash events is $\varepsilon = \{\boldsymbol{x}: Y(\boldsymbol{x}) < 0\}$. Using this

performance function, we can also define "crash events" to have a minimum range smaller than $b_{M-1} > 0$, i.e., $\varepsilon_{M-1} = \{\boldsymbol{x}: Y(\boldsymbol{x}) < b_{M-1}\}$. Following this procedure, a sequence of sets is constructed: $\varepsilon_M \equiv \varepsilon \subset \varepsilon_{M-1} \subset \cdots \subset \varepsilon_1 \subset \varepsilon_0 = \Omega$, and the corresponding performance criteria are$b_M \equiv b = 0 < b_{M-1} < \cdots < b_1 < b_0 = +\infty$.

Implementation details of SS, in particular, the choice of level probability p_m defined in Equation (3.19) and proposal distributions $q_k(\cdot \mid \cdot)$ for each dimension, are discussed in [111]. It has been confirmed that $P_m = 0.1$ proposed in the original paper [108], is nearly optimal. While the choice of the proposal distribution $q_k(\cdot \mid \cdot)$ in MMA is more delicate. Any one-dimensional distribution centered at the seed could suffice, but the shape of the distribution may affect the efficiency of the MMA in a non-trivial way: proposal $q_k(\cdot \mid \cdot)$ with both small and large variance tend to increase the correlation between successive samples, making statistical estimation of the conditional probability (level probability) $P_m = \mathbb{P}(\varepsilon_m|\varepsilon_{m-1})$ in Equation (3.17) less efficient. The most well-studied candidate distribution is the normal distribution, i.e. $q_k(\cdot\,|x_i) = N(x_i, \sigma_k)$, where x_i is the mean value and σ_k is the standard deviation. In [73], [108], [111], the optimal standard deviation was found to be related to the "roughness" I of a PDF f, defined as

$$I = \mathbb{E}_f[((\log f)')^2] = \int_{-\infty}^{+\infty} \frac{\left(f'(x)\right)^2}{f(x)}\, dx \tag{3.36}$$

which is given by $\sigma_k \approx 2.4/\sqrt{KI_k}$ in [111], where K is the dimension of original distribution and I_k is the "roughness" of the k^{th} dimension PDF.

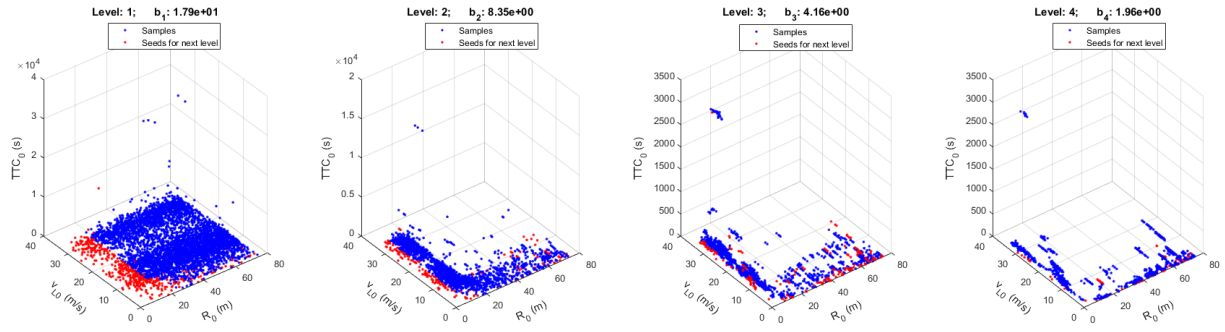
The SS parameters used in our simulations are summarized in Table 3.3. The variance of each proposal distribution is calculated accordingly. The total number of samples for each level is 5000, and the total number of states for each MCMC chain is 10. The simulation terminates when the performance criterion for the next level is smaller than 0 or the iteration level reaches 10. Our termination conditions mean that either the crash cases are found or the crash rate of the testing AV is as low as 10^{-10}.

Table 3.3 Parameters for the Subset Simulation

	3-variable Stochastic Model	BGGMM
Original distribution $f(\boldsymbol{x})$	$f_{BL}(v_{L0}, TTC_0, R_0)$ $= f_1(v_{L0}) f_2(TTC_0) f_3(R_0)$	$f_{BGGMM}\left(v_F^{(0)}, \boldsymbol{v_L}, R_0, T\right)$
Proposal distribution $q_k(\cdot \mid \cdot)$	$N(x_i \mid \sigma)$, where $\sigma = \begin{cases} 1, & x_i \sim f_1(v_{L0}) \\ 0.11, & x_i \sim f_2(TTC_0) \\ 0.003, & x_i \sim f_3(R_0) \end{cases}$	In space U (Equation (3.34)): $N(u_i \mid \sigma)$, $\sigma \approx 1.07$
Performance function $Y(\boldsymbol{x})$	min R during the entire lane change	
Level probability P_m	0.1 for each level m	
Total samples for each level N	5000	
Number of seeds for each level $N \times P_{m-1}$	$5000 \times 0.1 = 500$	
Total state for each Markov Chain N_c	$N_c = p_m^{-1} = 10$	
Stop criteria	$m + 1 > 10$ or $b_{m+1} < 0$	

3.4.3 Evaluation Results

The ACC+AEB system is tested by the initial condition sampled from the baseline 3-variable stochastic model. The three variables are the initial range, initial lane change vehicle's speed, and initial TTC. The samples tested using SS are shown in Figure 3.10, with the x-axis being the initial range, the y-axis being the initial lane change vehicle speed and the z-axis being the initial TTC. Only the first four levels are shown. The failure probability $P_F = 3.1 \times 10^{-7}$ is calculated by Equation (3.17) using only 32,000 test results from SS. This is less than half of the 74,100 cases needed by the IS technique [5], with the same c.o.v. value.

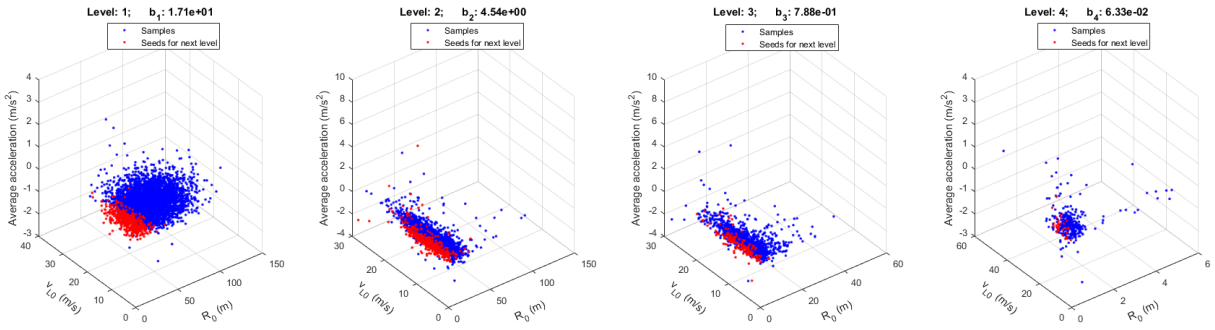


(a) Level 0 samples (b) Level 1 samples (c) Level 2 samples (d) Level 3 samples

Figure 3.10 Samples Expended by SS using the Baseline 3-variable Model

As shown in Figure 3.10, the test cases continue to converge to lower TTC values from level to level. The lower the TTC, the shorter time for the ACC+AEB to react to. The results also show that three clusters of risky cases emerge: (1) low initial range with various initial lane change vehicle's speed from 2m/s to 35m/s; (2) low initial lane change vehicle's speed with a range from 2m to 75m; (3) high initial TTC (around 2600s) with short initial range and high initial lane change vehicle's speed. Seeds in level 4 give dangerous cases that would happen with probability 10^{-4}.

The ACC+AEB system is tested using the environment model BGGMM. The samples tested at each level using SS are shown in Figure 3.11, with the x-axis being the initial range, the y-axis being the initial lead vehicle speed and the z-axis being the average acceleration of the lead vehicle during a lane change. The red points are those chosen to be the seeds for the next level. In total, 18,500 cases are tested, and in level 5, $b_5 < 0$, thus the simulation is terminated. Calculating from Equation (3.17), the failure probability is $P_F = 3.45 \times 10^{-4}$. It is much higher than using the baseline 3-variable stochastic model, which is $P_F = 3.1 \times 10^{-7}$. This is because during the lane change, the BGGMM enables the lane change vehicle to decelerate, which will endanger the following AV. The evaluation results are summarized in Table 3.4.



(a) Level 0 samples (b) Level 1 samples (c) Level 2 samples (d) Level 3 samples

Figure 3.11 Samples Expended by SS using the 8-variable BGGMM Model

Table 3.4 Evaluation Results Summary

	3-variable Stochastic Model		8-variable BGGMM	
	Subset Simulation	Importance Sampling	Subset Simulation	Importance Sampling
Crash Rate	3.1×10^{-7}	2.1×10^{-7}	3.45×10^{-4}	-
Total Tests	32,000	74,100	18,500	-

As can be seen in Figure 3.11 (d), the ACC+AEB performs poorly when the lead vehicle decelerates during a lane change and when the initial range is short. However, we do not see obvious clusters like in the baseline 3-variable stochastic model. The reason is that the SS only selects the top 10% of most dangerous cases at each level and when the lane change vehicle can decelerate, cases with lower initial range are more dangerous than other cases at the same probability level. Also, when the lane change vehicle's speed is between 20m/s (45mph) to 40m/s (89mph), it is more likely to decelerate during a lane change. Another factor to note is that Figure 3.11 only shows three dimensions.

3.5 Summary

In this chapter, we proposed the Subset Simulation (SS) as an adaptive sampling method for the accelerated evaluation of automated vehicles. The SS has two main advantages. First, SS can deal with black-box systems, i.e., no information about the AV control algorithm is needed. Moreover, as described in Section 3.2.2 , the Markov Chain Monte Carlo (MCMC) method is

used in SS (particularly MMA) to extend inside the variable space. This property enables SS to deal with high dimension stochastic models. These two advantages are very important when assessing AV's safety. We demonstrate the ability of SS to accelerate the evaluation in Section 3.4.3 . In general, SS is a variance reduction technique aiming to use fewer tests to estimate the performance and have better-accelerating performance than the importance sampling method (twice data-efficient than the importance sampling method).

In this chapter, an 8-variable BGGMM stochastic model is developed to describe the vehicle motions during lane changes. This model allows the leading vehicle to accelerate or decelerate during lane changes and is more comprehensive than the baseline 3-vairable stochastic model used in the literature.

The limitation of the SS method is that it cannot calculate the c.o.v. during the evaluation, thus not able to fix the number of testing. Moreover, the “danger regions” are searched as the test procedure unfolds. If the environmental statistics change, the crash rate cannot be estimated accurately.

Chapter 4 Evaluation of the Autonomous Vehicle's Policy Using Learning-Based Approach

The evaluation method proposed in the previous Chapter requires the knowledge of the environment statistics. However, environmental statistics information is not always available and may not be time-invariant. For instance, variations can be due to normal hours vs. rush hours, weather conditions, etc. In this chapter, we study the evaluation problem using another set of assumptions (the 2nd set of assumptions listed in Section 1.4) that we **do not have access to the environmental statistics**. Moreover, the prior developed subset simulation method cannot be used to test an active control system or a decision-making system. It lacks the ability to trigger the active motion. Therefore, in this chapter, we develop a learning-based evaluation method that can trigger the active lane change of the AV we designed in Chapter 2 and generate reasonable perturbation without the environment statistics.

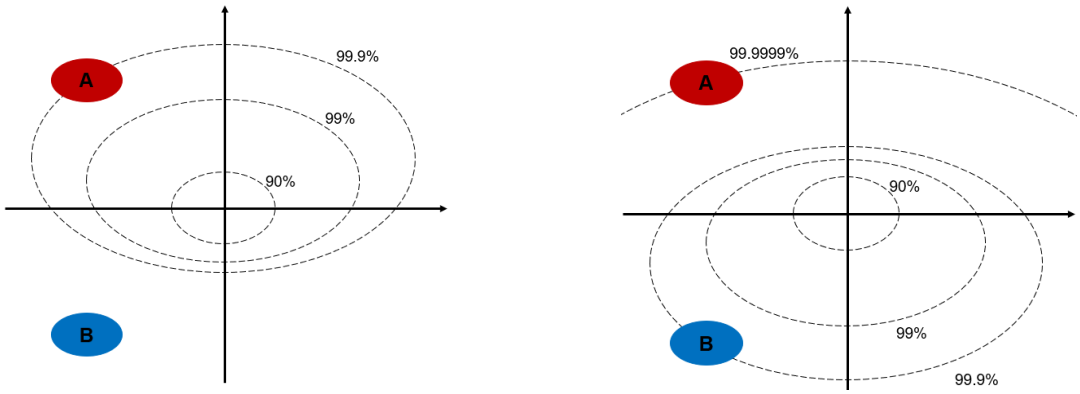
4.1 Motivations

4.1.1 Difficulties of implementing IS and SS with different environments

It is necessary to clarify the difficulties of implanting IS and SS methods when the environmental statistics are changing with time. If the IS or SS can be implemented across different environments, there is no need to develop approaches assuming not having access to environmental statistics.

For the IS method, assuming a black-box AV system, we need to search for the ideal ISD, i.e., the zero-variance distribution derived in Equation (3.10). However, the ideal ISD highly depends on the original environment distribution. Even if we managed to find the zero-variance distribution in one environment, its convergence performance is unknown in other environments. Searching for zero-variance distribution for a new environment is time-consuming and laborious, and convergence is not guaranteed if we are evaluating under several environments. This difficulty increases exponentially if the environment model is in high dimensions, which means it is necessary to implement **sequential IS** or **sequential MCMC**

(e.g., SS) method. To elaborate this problem, a schematic diagram is used. As illustrated in Figure 4.1, the system has two danger regions, A and B, and we assume not knowing any of them. Imaging, we first evaluate the system (using SS or other technique) in environment I (Figure 4.1 (A)), danger region A is revealed, and the crash rate is estimated to be $1 - 99.9\% = \%0.1$. Then we want to evaluate the system in environment II (Figure 4.1 (B)), and we know the distribution of environment II. If we estimate the crash rate use danger region A, without redoing the sequential search, the crash rate is calculated as $1 - 99.9999\% = 0.0001\%$. However, the true crash rate under environment II should be $1 - 99.9\% = 0.1\%$, since another danger region B should be revealed and dominate the result in environment II. Therefore, sequential IS or sequential MCMC need to be redone from the beginning, which makes it very time consuming if we are evaluating under several environments.



(a) Environment I (b) Environment II

Figure 4.1 Sequential Search of Danger Regions

Inspired by these reasons, we aim to find the set of failure cases ε directly. And then, given a specific environment, we can calculate the crash rate from the probability of the set of failure cases $P(\varepsilon)$ without additional tests.

4.1.2 Decision-making System Evaluation

As may have been noticed, the evaluation method we developed in Chapter 3 was applied to a level 2 AV control feature, i.e., the ACC+AEB system. It is easy to evaluate a passive or reactive control algorithm since, in this case, and the environment produces a disturbance, and

the system under testing reacts to the disturbance. However, when evaluating a decision-making system, we need to design tests to trigger the active motion of the AV and evaluate the consequence.

There are papers reporting the work on evaluating AV features other than ACC and AEB. In [50], [131], Tuncali et al. developed a simulation-based falsification method to generate the worst cases for a side collision avoidance system and a rear collision avoidance system. In [132], Tuncali et al. extended the method to evaluate an AV system with object detection components and tested it in a left-turn scenario. Formally verifying an AV algorithm's "correctness" requires that the "dangerous situations" are caused by other drivers. Moreover, it is hard to address the evaluation of deep learning-based systems, which makes formal verification methods intractable.

In [72], O'Kelly et al. trained a Deep Neural Network (DNN) environment model using the Generative Adversarial Imitation Learning (GAIL) method with highway driving data collected on I-80, California. For testing AV, an adaptive IS method is used to find the optimal ISD, and the test procedure is accelerated. Even though the DNN-based environment model can test high automation level features, it derives from a naturalistic driving database and thus run contrary to our assumption. Therefore, a more advanced method is needed.

4.2 Literature Reviews on Attacking Deep Neural Networks

The objective of the study in this chapter is to evaluate the decision-making system we developed in Chapter 2 without environmental statistics. Since the decision-making system we developed is trained by Deep Reinforcement Learning (DRL) and the policy is represented via a Deep Neural Network (DNN), the robustness analysis method for the traditional control system is not applicable. To develop an evaluation method that can test the DRL policy (trained in Chapter 2), we first did a thorough literature review on how to attack the DNN and DRL policy.

4.2.1 Attacks on Deep Neural Network

Over the past two decades, deep learning algorithms and deep natural networks (DNN) have been widely used in fields including image recognition and classification [133], speech recognition [134], and natural language processing [135]. Recent research shows that DNNs may be vulnerable to adversarial perturbations and attacks. In [136], the authors found that adversarial image patches can lead white-box DNNs to erroneous classification results. Papernot et al. in

[137] further developed an attack using synthetic data generation to craft adversarial image examples misclassified by black-box DNNs. In [138], the authors successfully fooled the YOLOv2 algorithm by sticking specially designed pattens to the human body; an example is shown in Figure 4.2. A more comprehensive overview of the adversarial attack on DNN can be found in [139].

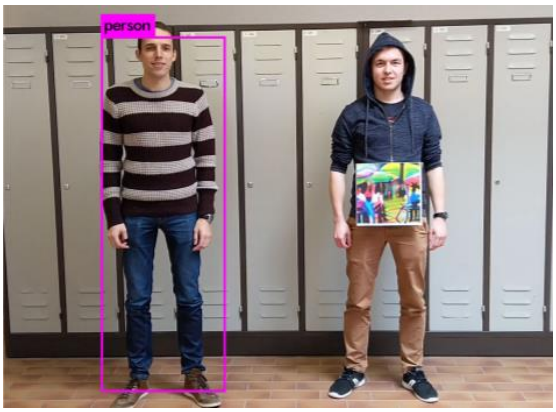


Figure 4.2 Generating Adversarial Patches against YOLOv2 [138]

4.2.2 Attacks on Deep Reinforcement Learning

DNNs have also been introduced in the field of deep reinforcement learning (DRL), where the goal is to train an agent to maximize the expected return. DNN works as an actor net or a critic function either to provide the optimal policy or to estimate the expected future return. Not surprisingly, the DRL policies are also vulnerable to adversarial perturbations. In [140], the authors characterize different types of attacks on DRL, as shown in Figure 4.3. DRL policies can be attacked by adding perturbation to observations, actions, or environment transition probabilities. To perturb observations, researchers first followed the same ideas as attacking DNN, which leads the DRL policy to use a different action [141], [142], or the observation can be modified directly or indirectly. For attacks applied to the action space, in [140] the author claimed that the action outputs could be modified by installing some hardware virus in the actuator executing the action. In [140], the environment transition model is also perturbed. However, as pointed out by the authors, these attacks are useful only under very specific conditions.

In the field of autonomous vehicles (AV), researchers also tried to attack existing AV systems for the purpose of evaluation or faster synthesis. In a recent report [143], Tencent's Keen Security Lab showed how they were able to bamboozle a Tesla Model S into switching lanes to drive directly into the oncoming traffic by manipulating the input video. This attack is in the category of observation attack. To our best knowledge, there is little study on attacking by varying the environment transition model, which is the focus of this chapter.

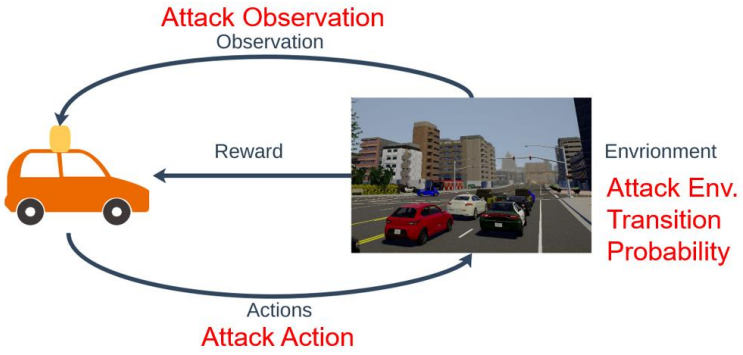


Figure 4.3 Attacks on Deep Reinforcement Learning

4.3 Methodology

As described before, the objective is to evaluate the decision-making system we developed in Chapter 2 without environmental statistics. To address this problem, we developed a simulation-based falsification method. We name **the AV** under testing **the victim** and model the victim as playing against an opponent (**the attacker**) in a two-player Markov game [144]. Our threat model assumes the attacker cannot directly control the victim.

As described in the literature reviews in Section 4.2.2 , DRL policy can be attacked by adding perturbations to observations, actions, or environment transition probabilities. As we assume our decision-making system has perfect observation and perfect actuator, we will focus on perturbing the environment transition probabilities to attack and evaluate the policy.

In this work, we model the victim training an attacker agent who will 'set up" the victim into crashes that are the victim's responsibility. The simulation-based falsification method we developed is based on a two-player Markov game [144].

4.3.1 Markov Game

A two-player Markov game can be denoted as $M = ((S_\alpha, S_\nu), (A_\alpha, A_\nu), P, (r_\alpha, r_\nu), \gamma)$, where we denote the attacker and victim by subscript α and ν respectively. It consists of the state set S_α and S_ν, action set A_α and A_ν, and a joint state transition function $P: S_\alpha \times S_\nu \times A_\alpha \times A_\nu \to \Delta(S)$ where $\Delta(S)$ is a probability distribution on S. The reward function $r_i: S_\alpha \times S_\nu \times A_\alpha \times A_\nu \to R$ for player $i \in \{\alpha, \nu\}$ depends on the current state, next state, and both player's actions. And γ is the discounted factor. Each player wishes to maximize their (discounted) sum of rewards.

The attacker is allowed unlimited black-box access to actions sampled from the victim's policy π_ν, but is not given any white-box information such as weights of its DNN function. We further assume the victim follows a fixed stochastic policy π_ν, corresponding to the common case of a pre-trained model deployed with static weights. Safety-critical systems are particularly likely to use a fixed or infrequently updated model due to the considerable expense of real-world testing.

Since the victim policy π_ν is held fixed, the two-player Markov game M reduces to a single-player MDP $M_\alpha = (S_\alpha, A_\alpha, P_\alpha, r_\alpha, \gamma)$ that the attacker must solve. The state space and action space of the attacker are the same as in M, while the transition and reward function has the victim policy π_ν embedded: $P_\alpha(s, a_\alpha) = P(s, a_\alpha, a_\nu)$ and $r'_\alpha(s, a_\alpha) = r_\alpha(s, a_\alpha, a_\nu)$, where the victim's action is sampled from the stochastic policy $a_\nu \sim \pi_\nu(\cdot \mid s)$. The goal of the attacker is to find an adversarial policy π_α maximizing the sum of discounted rewards:

$$\sum_{t=0}^{\infty} \gamma\, r_\alpha\left(s^{(t)}, a_\alpha^{(t)}, s^{(t+1)}\right), \text{ where } s^{(t+1)} \sim P_\alpha(s, a_\alpha) \text{ and } a_\alpha \sim \pi_\nu(\cdot \mid s) \qquad (4.1)$$

Note the MDP's dynamics P_α will be unknown even if the Markov game's dynamics P are known since the victim policy π_ν is a black-box policy. Consequently, the attacker must solve an RL problem.

In our application, the π_ν is what we learned in Chapter 2 and thus the $P_\alpha(s, a_\alpha) = P(s, a_\alpha, a_\nu)$ can be acquired by sampling $a_\nu \sim \pi_\nu$. And we can train the attacker to see the learned policy as part of the attacker's environment. Thus, the next question would be how we should define the reward function for the attacker? In the next section, we will first describe the attacker's training environment's state space and action space. And then, in the following sections, we will describe the reward function design for the attacker.

4.3.2 Attacker's Training Environment

To train the attacker, we first define the state space and the action space. The state-space includes information from up to 6 surrounding vehicles and the victim AV. As defined in Chapter 2, the information of each vehicle includes its relative longitudinal position, relative lateral position, and relative speed. In addition, we also include the victim AV's state and action in the attacker's state space. Therefore, in total, we have a 2 (the attacker's state) + 3 (surrounding vehicle's state) × 7 (cars) + 1 (the victim's action) = 24-dimension state space, i.e. $S \subseteq \mathbb{R}^{24}$. The states are scaled for efficient neural network training. The action space of the attacker is the same as the victim, which is described in Section 2.3.1 .

At initialization, the attacker is located near the victim AV. As shown in Figure 4.4, the attacker (the red box) can be observed by the victim AV (the blue box). Although the attacker can observe the state variable values, the attacker does not have direct access to the victim AV's policy. Moreover, the victim AV does not explicitly know which car is the attacker. Therefore, we are performing a black-box attack. The green boxes represent the surrounding vehicle that the attacker can observe.

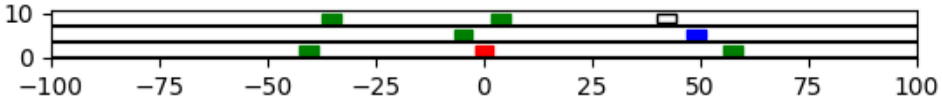


Figure 4.4 Attacker's Training Environment

4.3.3 Socially Acceptable Attacks

Before discussing the reward function design for the attacker, we would like to illustrate what is going on from the victim's perspective. In Chapter 2, we trained the victim policy, i.e., the discretionary lane change policy, using the reinforcement learning method. It results in giving us an optimal policy in a certain MDP environment. If now we add an attacker to the environment, the transition probability will be different, and the trained policy will no longer be optimal. Therefore, adding an attacker to the environment and perturbing the transition probability will definitely fail the victim. However, we may end up getting a "crazy" attacker

and getting useless results. Therefore, we need to constrain the perturbation by additional requirements.

A reasonable additional requirement is related to who is responsible for the crash. It is easy to deliberately ram into a victim AV by designing a "crazy" attacker, but coming up with reckless or unavoidable crash scenarios does not help to design a better AV. On the other hand, if the victim AV is lured into a situation and subsequent actions of the attacker result in a crash that is the responsibility of the victim AV according to common traffic rules, such an attack is useful for a subsequent redesign of the AV. We call this kind of attack the "socially acceptable attack."

The objective of this study is to design an attacker to generate "socially acceptable attacks" to the victim AV. In the rest of this chapter, the policy trained in Chapter 2 is under test and denoted by "the victim AV," while the training agent is "the attacker."

4.3.4 Reward Considering Socially Acceptable Attack

The key component for training an attacker to generate Socially Acceptable Attacks (SAAs) is the reward. The attacker is rewarded if it causes a collision between the victim AV and one of the surrounding cars (not necessarily be the attacker), in which the victim AV is at fault. In this work, we use the ideas from Responsibility-Sensitive Safety (RSS) [145] model and encode the traffic rules through associated rewards to train the attacker. First, we recall the 5 "common sense" rules followed by RSS:

1. Do not hit someone from behind.
2. Do not cut-in recklessly.
3. Right-of-way is given, not taken.
4. Be careful of areas with limited visibility.
5. If you can avoid an accident without causing another one, you must do it.

The first three "common sense" principles above are related to traffic rules and can be implemented through associating rewards with the pre-crash state. The fourth is not applicable in our application since we assume perfect perception. To implement the fifth, we refer to another related paper from Shashua et al. [146]. They implemented the RSS model on NHTSA pre-crash scenarios, where they define "proper response" to dangerous situations (related to the fifth rule) as using Minimal Evasive Effort (MEE). MEE deals with cases in which extra caution is applied to prevent potential situations in which responsibility might be shared. Here we develop a similar

concept, the Best Evasive Effort (BEE), to the responsible vehicle, which defines the best action the responsible vehicle can take to avoid the crash. To explain, let us assume the victim AV is the responsible vehicle, and we find an SAA resulting in the victim AV crashing into its front vehicle without doing the BEE (i.e., hard brake). Then, this SAA should be reward more since it finds a fatal case of the victim AV.

In reinforcement learning, the reward function is $r(s, a)$, where the s is the state and a is the attacker's action at state s. To clearly define the responsibility of a crash, we predict the situation at the next time-step with the victim AV's action and the attacker's action at the current time-step, getting the next state s'. Since the simulator is deterministic, the reward function can be further extended to $r(s, a, s')$. If at the next state s' the victim AV crash, then the s is the pre-crash state.

In this work, instead of implementing the whole RSS model [145], which considers an entire pre-crash scenario, we only consider one pre-crash state. Therefore, no blame time concept as in [146] is implemented in this application. In the future, if one wants to implement the RSS model considering the entire scenario, we recommend modeling the attacker's policy with a recurrent neural network and design the reward function accordingly.

The BEE is implemented in the reward corresponding to each pre-crash situation. As only one timestep is being considered and the action space is discrete, we define the right choice of action as BEE for each pre-crash state. In this work, we only define the BEE for the responsible car for simplicity, but the BEE for the irresponsible car can also be defined similarly. In the future, if one wants to implement the BEE concept in the environment with continuous action space, the BEE should be calculated according to the RSS model, as conducted in [146]. The pre-crash state can be categorized as 1) no car is on the lane marker; 2) only one car is on the lane marker; 3) both cars are on the lane marker. For each pre-crash state, we separate the responsibility and define the choice of BEE, then assign reward accordingly.

As shown in Figure 4.5, when **no car is on the lane marker**, the rear car (the blue one) is always responsible when a crash happens between these two vehicles. And the right choice of action as BEE for the rear car in this pre-crash state is the **hard brake**. If the rear car is the victim AV and the action of it is not the BEE (i.e., hard brake), then the reward for the attacker is 1, since the attacker triggered a fatal failure of the victim. If the rear car is the victim AV and the action of it is the BEE (i.e., hard brake), then the reward for the attacker is 0.5. While if the rear

car is not the victim AV, the attacker will be punished. The reward for no car is on the lane marker case is summarized in the first row of Table 4.1.

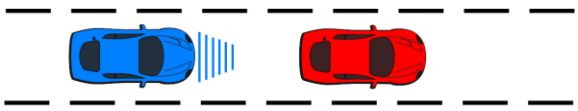


Figure 4.5 No Car on the Lane Marker

In the second type of pre-crash state, where there is **only one car on the lane marker**, the crash can happen between the lane change car and the car in the original lane or between the lane change car and the car in the target lane. When the car on the lane marker crashes with the car in the original lane, the reward is designed similar to the situation when no car is on the lane marker (as described above). Figure 4.6 shows the other situation, in which only one car is on the lane marker, and it crashes with the car in the target lane. In this case, the lane change car (the blue one) is responsible if a crash happens. Then the BEE of the responsible car should be **abandoning the lane change**. If the lane change car is the victim AV and the action of it is not the BEE (i.e., abandoning the lane change), then the reward for the attacker is 1, since the attacker triggered a fatal failure of the victim. If the lane change car is the victim AV and the action of it is the BEE (i.e., abandoning the lane change), then the reward for the attacker is 0.5. While if the lane change car is not the victim AV, the attacker will be punished. The reward for only one car is on the lane marker case is summarized in the second and third row of Table 4.1.

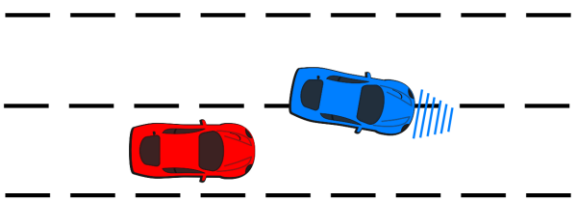


Figure 4.6 Only One Car on the Lane Marker and Crash with the Car in the Target Lane

In the third type of pre-crash state, where **both cars are on the lane marker**, they can on the same lane marker or on different lane markers. When both cars are on the same lane marker,

the reward is designed as analogous to the situation when no car is on the lane marker. When cars are on different lane markers, as shown in Figure 4.7, according to multiple traffic laws [147]–[149], both cars are at fault. According to the Chinese traffic law [147], the vehicle from the left lane should yield to the vehicle from the right lane. While in the Texas traffic law [148], the vehicle from the right lane should yield. And the right-of-way is not clarified in the New York traffic law [149]. Here we take the Chinese traffic law as an example, i.e., the left car (blue car) is mainly responsible for the collision and is expected to abort the lane change to avoid a crash. Therefore, the BEE of the main responsible car should be **abandoning the lane change**. If the mainly responsible car is the victim AV and the action of it is not the BEE (i.e., abandoning the lane change), then the reward for the attacker is 0.8. If the mainly responsible car is the victim AV and the action of it is the BEE (i.e., abandoning the lane change), then the reward for the attacker is 0.2. While if the mainly responsible car is not the victim AV, the attacker will be punished. The reward for the case when both cars are on the lane marker is summarized in the 4th and 5th row of Table 4.1.

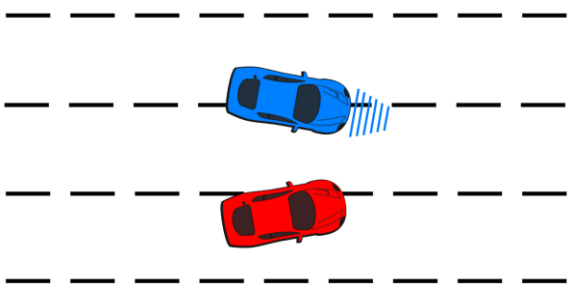


Figure 4.7 Both Cars are on the Different Lane Markers

The attacker also has a time cost of -0.05 per step, which encourages the attacker to cause the victim AV to collide as soon as possible. An episode is terminated if either of the following happens:

1. The victim AV crashes with another car, and the reward is given to the attacker according to the pre-crash state of the victim AV (as in Table 4.1).
2. The attacker crashes with a car other than the victim AV, and the reward for the attacker is -1.

3. The AV leaves the neighborhood of the attacker, i.e., not being one of the six surrounding cars of the attacker, and the reward is -1.

We also record the Failure Code (FC) for each episode, as shown in Table 4.1. The definition of each code is described as follows:

0: The **other car is responsible** for the crash, and the action of it **is not** the BEE.

1: The **other car is responsible** for the crash, and the action of it **is** the BEE.

2: The AV crashes into the **front vehicle without** a hard brake (BEE in this case).

3: The AV crashes into the **front vehicle with** a hard brake (BEE in this case).

4: The AV changes lanes and crashes with the **target lane vehicle without** BEE.

5: The AV changes lanes and crashes with the **target lane vehicle with** BEE.

6: The AV changes lanes from the left to the middle lane and crashes with the car **changing lane from the right to the middle lane**, and its action **is not** BEE.

7: The victim changes the lane from the left to the middle lane and crashes with the car **changing lane from the right to the middle lane**, and its action **is** BEE.

Table 4.1 Reward Design for Responsibility-sensitive Attack

<table>
<tr><th rowspan="2">Pre-crash situation</th><th rowspan="2">Responsible car</th><th colspan="2">Best Evasive Effort (BEE)</th><th colspan="3">Attacker's reward</th><th rowspan="2">FC</th></tr>
<tr><th>Responsible car</th><th>The other car</th><th>Fault</th><th>BEE</th><th>Reward</th></tr>
<tr><td rowspan="4">No car is on the lane marker</td><td rowspan="4">The rear car</td><td rowspan="4">Hard brake</td><td rowspan="4">-</td><td rowspan="2">Not the Victim</td><td>No</td><td>-1</td><td>0</td></tr>
<tr><td>Yes</td><td>-0.5</td><td>1</td></tr>
<tr><td rowspan="2">Victim AV</td><td>No</td><td>1</td><td>2</td></tr>
<tr><td>Yes</td><td>0.5</td><td>3</td></tr>
<tr><td colspan="8">Only one car is on the lane marker and crashes with the car in the original lane: Same as no car is on the lane marker</td></tr>
<tr><td rowspan="4">Only one car is on the lane marker: crash with the car in the target lane</td><td rowspan="4">The lane change car</td><td rowspan="4">Abandon the lane change</td><td rowspan="4">-</td><td rowspan="2">Not the Victim</td><td>No</td><td>-1</td><td>0</td></tr>
<tr><td>Yes</td><td>-0.5</td><td>1</td></tr>
<tr><td rowspan="2">Victim AV</td><td>No</td><td>1</td><td>4</td></tr>
<tr><td>Yes</td><td>0.5</td><td>5</td></tr>
<tr><td colspan="8">Both cars are on the same lane marker: Same as no car is on the lane marker</td></tr>
<tr><td rowspan="4">Both cars are on the lane marker: different lane markers</td><td rowspan="4">Shared fault: but the left car is of the principal fault</td><td rowspan="4">Abandon the lane change</td><td rowspan="4">-</td><td rowspan="2">Not the Victim</td><td>No</td><td>-0.8</td><td>0</td></tr>
<tr><td>Yes</td><td>-0.3</td><td>1</td></tr>
<tr><td rowspan="2">Victim AV</td><td>No</td><td>0.8</td><td>6</td></tr>
<tr><td>Yes</td><td>0.2</td><td>7</td></tr>
</table>

As shown inTable 4.1, the AV-responsible crashes, in which the action of the victim AV is not BEE, are valued most by the attacker (i.e., Failure Code: 2, 4, and 6). This kind of crash is the most deadly crash. Moreover, the action chosen by the victim AV's policy before these crashes deserves a closer look and may require revision. In summary, instead of finding a crazy attacker, we trained an attacker to generate “socially acceptable attacks,” which explores the weakness of the victim AV and helps to improve its policy.

4.4 Training Setups

4.4.1 The Victim AV under test

We study a discretionary lane change decision-making problem in this chapter. The state space, action space, the victim training reward, and the simulation environment are introduced in Chapter 2. We evaluate exactly the same lane change decision-making policy that we trained in Chapter 2 for comparison. This victim AV agent will be evaluated by both the SAPs and in the original environment in this chapter.

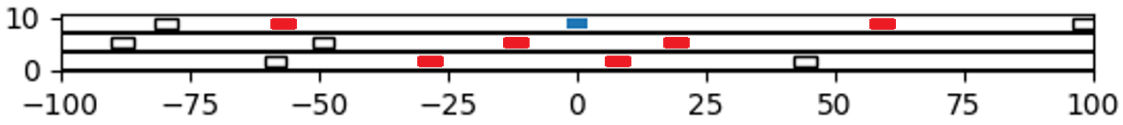


Figure 4.8 Three lane highway simulator. The blue box: the AV; red boxes: 6 nearest surrounding vehicles; empty boxes: unobserved surrounding vehicles

Here is a brief refresher of the original victim AV policy. The driving environment used to train the victim AV in this work is a three-lane highway simulator. The AV is driving with up to six surrounding vehicles (three vehicles in front, three vehicles behind), as shown in Figure 4.8. The blue box is the AV, and the six red boxes are the six surrounding vehicles whose states are observed. The remaining boxes are environment vehicles whose states are not observed by the victim AV. The surrounding vehicles’ driving strategy is also described in Chapter 2.

Additional to the policy, we also implement the short-horizon safety check we described in Section 2.3.5 The short-horizon safety check method will replace dangerous action before

applying it. If the action chosen by the victim AV is unsafe, it will be replaced by IDM and AEB system-based action. However, as will be shown in Section 4.4.2 , this safety check cannot cover all the situations. The attacker can still find a way to fail the victim AV with an AV-responsible crash.

4.4.2 Training Setups for the Attacker

In this section, the simulation setup for training the attacker is described. The reinforcement learning algorithm we use is DDQN. The hyper-parameters used during training is shown in Table 4.2. To accelerate the training and achieve better exploration, we use two replay buffers, one for storing trajectories without any crash or with AV-irresponsible crash and the other for storing AV-responsible crash cases, while implementing the model-based exploration method described previously in Chapter 2. The reward for socially acceptable attacks and the failure code is a way to prioritize the collected samples. The use of two replay buffers is a simpler version of prioritized experience replay as discussed in [150], which has been implemented in [24]. The model-based exploration method presented in Chapter 2 can help the attacker to explore the weakness of the victim AV based on its understanding of the victim AV's policy.

Table 4.2 Hyperparameters for Training the Attacker

	Description	Value
γ	Discount factor	0.9
Δt	Sampling time	0.1 sec
ρ	Learning rate	1×10^{-6}
ε_0	Starting value for ε-greedy exploration	0.2
C	Annealing factor for ε-greedy exploration	2×10^{-6}
T	Steps for each episode	200
E	Total training episode	1×10^{5}

After training the attacker, the victim AV is evaluated in the original environment (without the attacker) and then the same environment with only one attacker. The AV is evaluated in both environments, with the total number of cars being 10, 15, and 20. The AV is

evaluated for 1×10^6 episodes, and each episode lasts for 200 steps unless terminated early due to a crash.

4.5 Attacker's Training Results and The Victim AV Evaluation Results

In this section, both the training curve of the attacker and the evaluation results of the victim AV in different environments are reported. Then, the attacker and the evaluation results will be used to improve the policy design of the original victim AV in Chapter 5.

As shown in Figure 4.9, the attacker is trained for 1×10^5 episode and evaluated by ten roll-outs every 100 episodes to be sure the attacker's policy is trained properly. We train the attacker ten times, starting with random initial DNN parameters and average the total rewards of the evaluation rollouts (shaded area represents mean ± standard error). As can be seen from Figure 4.9, the attacker's policy start to converge after 4×10^4 training episodes, which means the attacker is ready to generate socially acceptable attacks to the victim AV.

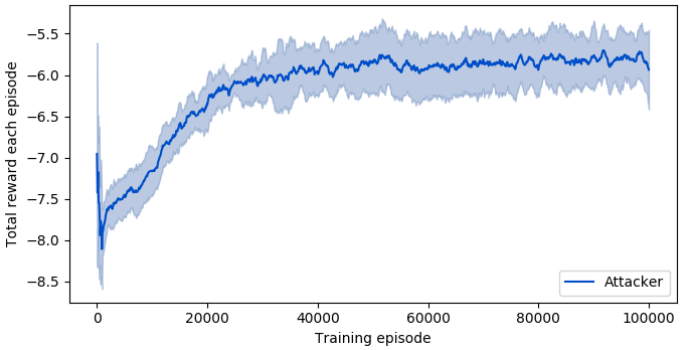


Figure 4.9 Training Curve of the Attacker

The victim AV is then evaluated in the original environment and the environment with this trained attacker. The detailed crash results are shown in Table 4.3. We have summarized the number of crash cases with different Failure Code (FC), respectively. The definition of FC is described in Section 4.3.4 . And here, we only focus on the victim AV's responsible crashes (FC 1 to 7). In each environment (with or without the attacker), we evaluate the victim AV with different numbers of total environment vehicles for a more comprehensive evaluation. In the bottom rows of Table 4.3, we report both the number of crashes between the attacker and the

victim AV (the first number), and the total number of crashes of the victim AV (the second number).

Table 4.3 Number of Crashes during Evaluation

# of env. cars		Failure Code							Crash rate
		1	2	3	4	5	6	7	
w/o. attacker	10	0	0	5	0	46	0	0	5.1×10^{-5}
	15	0	0	14	0	55	0	0	6.9×10^{-5}
	20	0	0	12	0	33	0	0	4.5×10^{-5}
w. attacker	10	656/657	**239/239**	180/186	**43/78**	101/156	**504/504**	9/9	1.8×10^{-3}
	15	447/448	**172/172**	163/164	**26/45**	63/88	**283/283**	3/3	1.2×10^{-3}
	20	419/598	**168/168**	137/139	**20/32**	59/64	**237/273**	1/1	1.1×10^{-3}

As can be seen from Table 4.3, there are many more crashes when one attacker is introduced. For crashes with Failure Code (FC) 2, 4, and 6, where the victim is liable, and the result crashes are fatal, the number of crashes jumps from 0 to hundreds. The total crash rates also increased around 50 times when including one attacker in the environment. This is a solid indication that our proposed method can generate Socially Acceptable Attacks (SAAs) by training an attacker, which provides useful insight into how the AV policy can be further improved.

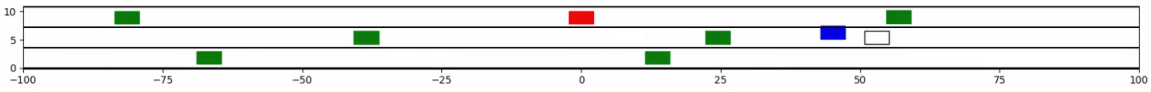


Figure 4.10 An example of the AV-responsible crash.

In Figure 4.10, we show one example of the AV-responsible crash. The victim AV (blue car), instead of braking and crashed by the following attacker (red car), changed lane and crashed into the target lane front car result in an AV-responsible crash. There are many similar crashes, and these crashes indicate the flaws in the original victim AV reward design described in Section 2.3.1 . The original victim AV will be punished -200 if there is a crash, no matter whose responsibility is the crash. Therefore, the trained victim AV from Chapter 2 does not learn to

separate responsibility. Moreover, as can be seen from the animation results, the victim AV does not know how to drive around the attacker. The victim AV sometimes will stop changing lanes in the middle of a lane change when the surrounding attacker also changes lanes. This indicates the original victim AV training environment does not contain enough diverse surrounding drivers and the trained DNN policy is not robust to SAAs. In the next chapter, we will solve the robustness problem of the trained DNN policy using the attacker we developed in this chapter.

4.6 Summary

In this chapter, we show that the AV policy learned by deep reinforcement learning (in Chapter 2) can be fragile, i.e., can still result in collisions even when the vehicles around it behave in a socially acceptable fashion.

We first design an attacker under the two-player Markov game framework to challenge the AV. The attacker can perceive the AV's behavior only through observations, and thus it tries to attack a black-box system. The attacker is trained to generate socially acceptable attacks by well designing the reward function that considers the accident's responsibility. The responsibility is separated by the pre-crash scenario, and if the attacker can lure the victim AV end up into an AV-responsible crash, it will be rewarded more. After the attacker's policy is trained to converge, the victim AV is evaluated in the environment with one attacker.

The evaluation results show that the attacker can lure the AV into many AV-responsible crashes, and the average crash rate increases 50 times more than the crash rate in the original training environment. The attacker will be further used to improve the robustness of the AV policy in Chapter 5.

Chapter 5 Meta Reinforcement Learning for Synthesis Adaptive Decision-making Policy

In Chapter 4, the trained AV policy is tested in both the original environment and the environment with one attacker. The results show that the AV policy is not robust towards the perturbation in the environment transition probabilities. In this chapter, we will focus on design a robust DLC policy for a wide range of MDPs with different transition probabilities. The trained policy will perform safely and efficiently not only in the original environment and the environment with one attacker but also perform safely and efficiently in unseen environments after adapting with limited data.

5.1 Motivation and Objective

In the previous Chapters, we first designed an AV agent using the state-of-the-art RL method, which can travel in the designed highway environment safely and efficiently. However, when it is driving in an environment with a different transition probability function, the crash rate soars. We have shown that, on the one hand, the Reinforcement Learning (RL) methods are powerful. It can solve the Markov Decision Process (MDP) problems with high-dimension state space. While on the other hand, it seems like the RL methods are useless to real-world applications.

The reason behind such observation is that the DNN is fragile, and DRL use DNN to represent its policy. Although DRL can find the optimal policy with respect to one MDP, it can fail in a similar MDP. Researchers have developed approaches to analyze the robustness of a DNN. In [151], the authors evaluate the robustness of a DNN with respect to input with geometric transformations in a worst-case regime and propose a new adversarial training scheme to improve the invariance properties of DNNs. And in [152], Liu et al. explicitly analyze the datapaths of the noise to improve the robustness of the DNN under noise. There are also some studies related to improving the robustness of DRL policy. In [153], [154], the authors try to adversarially perturb the simulation by choosing parameters that determine a simulation's dynamics, such as mass, the center of gravity, or friction during training. This method is called

the domain randomization method. The learned policies are more robust to a distribution shift in the underlying physics from simulation to real-world [153], [154]. However, the studies only consider the situations which have perturbations on the underlying physics models. And they are model-based methods. Another kind of approach is the verification method based on a formal method that finds theoretically proven bounds on the maximum output deviation, given a bounded input perturbation [155]–[157]. The difficulties in this approach are from the non-liner activation functions in the DNN. Also, it is computationally inefficient and can be inapplicable to some complicated problems. Therefore, a more efficient and generic approach is necessary.

Moreover, in our application, we are dealing with multi-MDPs environments with respect to the transition probabilities. Drivers in different locations [123], at a different time [158], or in different weather conditions [158] behave differently. It is necessary to solve this large variability in driving behavior by designing an adaptive system that can recognize the condition (implicitly) and adapt its policy accordingly.

By these motivations, we derive our objectives. The objective of this chapter is to develop the training and adapting approach that can teach the policy to adapt to different unseen environments with limited data quickly. The environments are different with respect to the transition probabilities. And we assume having the distribution of the environments.

5.2 Literature Review and Meta Reinforcement Learning Preliminary

5.2.1 Mathematical Formulation

First, we give the mathematical formulation of the previously described problem. In Chapter 2, we were trying to solve a single MDP problem, and the target is to solve for:

$$\alpha = argmax_{\alpha} E_{\pi_{\alpha},M}\left[\sum_{t=0}^{\infty}\gamma^{t} r_{t}\right] = f_{RL}(M) \tag{5.1}$$

where the α is the parameter vector of policy π, r is the reward function, and γ is the discount factor. It can be written as a function of the MDP M, i.e. $f_{RL}(M)$. To solve such a problem, we have introduced some algorithm in Section 2.2 .

Now, instead of solving a single MDP problem, we are trying to solve the multi-MDPs adaptation problem in the form:

$$\theta = argmax_\theta \sum_{i=1}^{N} E_{\pi_{\phi_i, M_i}} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \tag{5.2}$$

$$\phi_i = f_\theta(M_i) \tag{5.3}$$

where the θ is the parameter of the *adaptation function* $f_\theta(M_i)$, and M_i is the i^{th} MDP environment $M_i = (S_i, A_i, P_i, r_i, \gamma_i)$. From Equation (5.2), we are learning an agent that after the adaptation, it can perform well in a set of environments (in total N environments).

The terms used in the multi-MDPs adaptation problem for numerically solving Equation (5.2) and (5.3) are the Outer loop/meta optimization and the Inner loop/adaptation. In [159], the authors provide a good schematic of them. As shown in Figure 5.1, the outer loop trains the parameter weights θ, which determine the policy of the inner-loop agent (part of its parameters or its parameters initialization). The inner loop agent interacts with a given environment for some episodes. For every iteration of the outer loop, a new environment is sampled from a *distribution of environments*, which share some common structure.

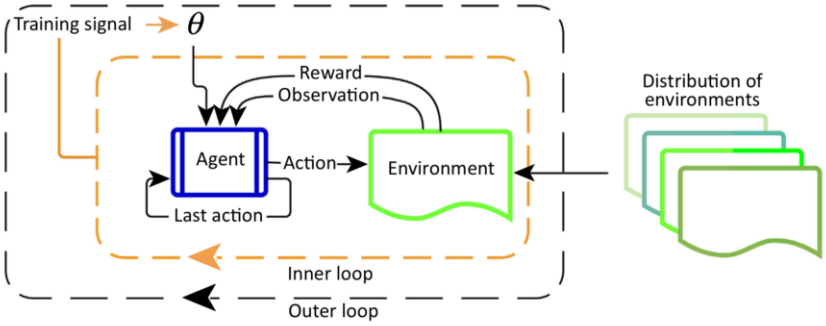


Figure 5.1 Illustrating the Inner and Outer Loops of Training [159]

5.2.2 Literature Review

Approaches for solving the multi-MDPs adaptation problem can be categorized by its meta optimization and adaptation steps. In the first kind of approach, the agent can learn to identify the environment by training an identifier using supervised learning. And then, for each model, the MPC or the DP method can be used to learn the policy. While during adaptation, we

can identify the model and switch to the corresponding controller. For instance, in [160], Nagabandi et al. use meta learning to train a dynamics model prior. And this prior can be rapidly adapted to the local context when combined with recent data. And the controller is extracted using model predictive path integral control. However, we need to enumerate the model with its structure, limiting the agent's generalization ability.

In other studies, researchers use behavior cloning to make the adaptation. For example, in [161], [162] Yu et al. present Domain-Adaptive Meta-Learning, a system that allows robots to learn from a single video of a human via prior meta-training data collected from related tasks. During training, the agent is provided with demonstration data. And we teach the agent how to infer a policy from one demonstration. And during testing, we provide an expert demo, and the agent runs behavior cloning. However, we don't assume to have a demonstration for the behavior cloning step.

There are also some model-free Meta Reinforcement Learning (MRL) that can solve the adaptation problem. In [163]–[165] the authors use a Recurrent Neural Network (RNN) to encode the MDP's information as the hidden memory of the RNN. And the policy contains that information in its weight to adapt to the different environments. However, there is no mathematical convergence proof for the RNN-based MRL policies. We cannot guarantee that the RNN-based MRL methods converge to a good adaptation function or even converge at all. Therefore, a more consistent MRL method is needed.

Another class of MRL methods uses the policy gradient approach for both the meta training and adaptation step [166]–[168]. In [166], Finn et al. developed the famous Model Agnostic Meta-Learning (MAML) method. The idea is that the agent is trying to find the parameter θ, such that when the agent takes a few gradient steps on that θ, it will get to a θ^* which is optimal for a given MDP. However, policy gradient method suffers in sparse reward environments. The agent cannot update its policy using trajectories with no reward.

Moreover, both the RNN-based and gradient-based approaches use on-policy RL methods (introduced in Section 2.2.2) for both the meta training and adaptation step and thus are data inefficient. The adaptation step is inherently on-policy learning since, given a new environment, the agent needs to collect new data using the current policy. However, the meta training step is not necessarily being on-policy learning. Therefore, Rakelly et al. [169] developed an off-policy meta training step based on the Soft Actor-Critic (SAC) [170] RL

approach and the stochastic encoder as the adaptor. The developed method is called the Probabilistic Embeddings for Actor-critic RL (PEARL). The PEARL MRL method is consistent, data-efficient, and has an advanced exploration strategy, being the state-of-the-art MRL approach. Therefore, we implemented this method in our application. In the next section, the details of PEARL will be expounded.

5.3 Efficient Off-policy Meta Reinforcement Learning Method

Solving the multi-MDPs adaptation problem can also be viewed as solving a Partially Observed Markov Decision Processes (POMDPs) problem. In a POMDP, the agent's decision processed is modeled assuming the system dynamics are an MDP, but the agent cannot directly observe the underlying state. Suppose each environment or task is modeled as an MDP, then an agent is trying to find the optimal policy without fully observe the hidden state (h_t in Figure 5.2).

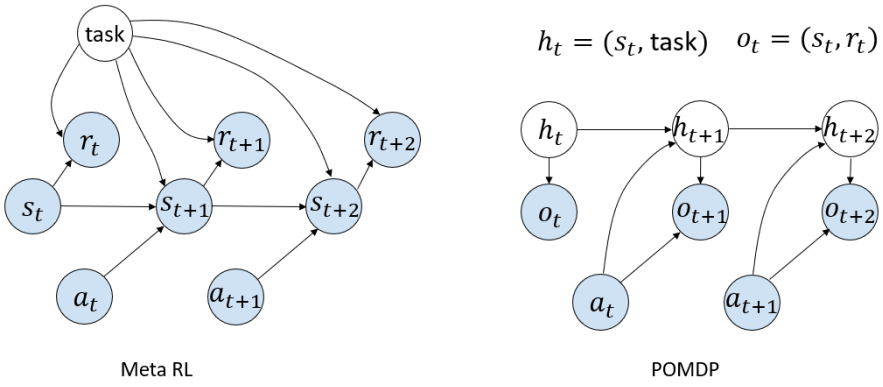


Figure 5.2 Multi-MDPs adaptation problem as a POMDP problem

To solve a POMDP problem, we can either directly solve the policy or explicitly estimate the hidden state. The PEARL [169] MLR method build on the second way. The agent learns from the experience collected in past environments to approximate its belief on a given environment. And to adapt, the agent will condition its policy on that belief. To capture uncertainty in the belief over the task, the agent learns a probabilistic latent representation of prior experience. Moreover, to achieve better data efficiency during meta-trading, the off-policy RL is implemented. In Section 5.3.1 , the model of the probabilistic latent representation of prior

experience is introduced. And then, in Section 5.3.2 , the detail of the adaptation step is elaborated. Finally, in Section 5.3.3 , we will explain the off-policy MRL method for the meta-training step.

5.3.1 Learning and Modeling Latent Contexts

First of all, the environments' information should be encoded by the latent context z to enable fast adaptation. In PEARL [169], a variational inference approach is implemented. The latent belief z is inferred by an inference netwok $q_\phi(z|c)$ given the context c (set of collected data), parameterized by ϕ which estimate the true posterior distribution $p(z|c)$. Typically, the objective is to optimize $q_\phi(z|c)$ to reconstruct the environment by learning the reward and transition function of the MDP. However, this is a model-based approach, and here we do not assume knowing the structure of the MDP reward and transition function. In a model-free manner, $q_\phi(z|c)$ can be optimized the maximize returns through the policy over the distribution of tasks. Therefore, the variational lower bound of the objective function is:

$$\mathrm{E}_{T\sim f(T)}\left[\mathrm{E}_{z\sim q_\phi\left(z|c^T\right)}\left[R(T,z)+\beta\times KL\left(q_\phi(z|c^T)||p(z)\right)\right]\right] \tag{5.4}$$

where the T is the current task or environment, $f(T)$ is the distribution that each task sampled from, $p(z)$ is a standard normal distribution $N(0,I)$ over z and the $R(T,z)$ can be a variety of objectives in task T at timestep t. We can understand the KL divergence term as a variational approximation to an information bottleneck that constrains the mutual information between z and context c. Intuitively, this bottleneck constrains z to contain only information from the context that is necessary to adapt to the task at hand, mitigating overfitting to training tasks [169]. This technique is also implemented in Section 2.3.3 The parameter ϕ will be learned though meta-learning, and during adaptation, the inference network will be used to infer the belief z from collected trajectories.

The inference network architecture design should be expressive enough to capture sufficient statistics of task-relevant information without modeling irrelevant information. Recall that encoding of a fully observed MDP should be permutation invariant. If we want to infer what the task is or identify the MDP model, it is enough to have access to a collection of transitions (s,a,s',r) without regard for the order in which these transitions were observed. Therefore, in

PEARL, the $q_\varphi(z|c_{1:N})$ network is modeled as permutation-invariant representation, i.e., as a product of independent factors:

$$q_\varphi(z|c_{1:N}) \propto \prod_{n=1}^{N} \Psi_\phi(z|c_n) \tag{5.5}$$

where the $\Psi_\phi(z|c_n) = N(f_\phi^\mu(c_n), f_\phi^\sigma(c_n))$, which results in a Gaussian posterior so that the method is tractable. The f_ϕ represent a DNN that predicts the mean μ and variance σ.

5.3.2 Advanced Exploration via Posterior Sampling

Modeling the latent context as probabilistic allows the agent to use the posterior sampling method for efficient exploration at adaptation time. In PEARL [169], the agent directly infers a posterior $q_\phi(z|c)$ over the latent context c (e.g., set of transitions (s, a, s', r)), which encode a specific MDP's information. We choose to encode the value function in our application since the backbone RL method we use is based on the value function. The meta-training procedure leverages training MDPs to learn a prior over z ($q_\phi(z)$) that captures the distribution over MDPs and learns how to use the experience to guide exploration. At adaptation time, the belief z is sampled from the prior ($q_\phi(z)$) first and the adapted policy is executed give the sampled z for an episode, thus exploring in a temporally extended and diverse manner. The agent can then use the collected experience to update our posterior and continue exploring coherently in a manner that acts more and more optimally as our belief narrows, akin to posterior sampling.

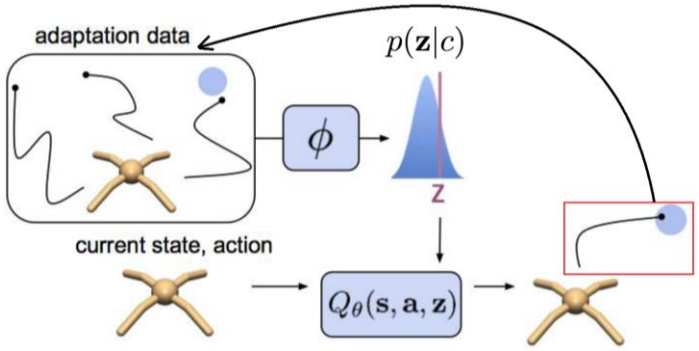


Figure 5.3 Collected experience can then be used to update the belief during adaptation [169]

5.3.3 Off-Policy Meta-Reinforcement Learning

In this section, we explain the meta-training procedure. The inefficiency of the meta-training process is largely elaborated in prior works [166]–[168], which is because of using stabel but relatively inefficient on-policy algorithms. However, implementing the off-policy RL method in meta-RL algorithms is non-trivial. During adaptation, the agent needs to collect new data in the new environment using the current policy, which is inherently on-policy data. And since at adaptation time, on-policy data will be used to adapt, on-policy data should be used during meta-training to train the encoder.

To solve this problem, we separate the data used to train the encoder from the data used to train the policy. The policy can treat the context z as part of the state in an off-policy RL loop, while the uncertainty in the encoder provides the stochasticity of the exploration process $q_\phi(z|c)$. The actor and critic are always trained with off-policy data sampled from the entire replay buffer B. We define a sampler S_c to sample on-policy context batches for training the encoder. By doing so, we have a separate off-policy meta-training and an on-policy adaptation step. The meta-training process of the PEARL algorithm is summarized in Algorithm 5.1.

Algorithm 5.1: PEARL Meta Training Algorithm

Initialization:
    Batch of training tasks $\{T_i\}$ from task distribution $f(T)$
    Replay buffers: B_i for each training task
for each training episode **do**
    for each T_i **do**
        Initialized context $c_i = \{\}$
        for $k = 1 \ldots \ldots K$ **do**
            Sample belief z from inference network $q_\phi(z|c_i)$
            Gather data using current policy $\pi_\theta(a|s,z)$ and add to B_i
            Update $c_i = \{(s_j, a_j, s_{j+1}, r_j)\}_{j:1,\ldots,N} \sim B_i$
    for step in training steps **do**
        for each T_i **do**
            Sample context using on-policy sampler, i.e. $c_i \sim S_c(B_i)$
            Sample RL off-policy training data $b_i \sim B_i$
            Sample $z \sim q_\phi(z|c_i)$
            Calculate loss function for the actor network L^i_{actor}, critic network L^i_{critic} and the stochastic encoder network L^i_{SE}
        Update actor network, critic network and stochastic encoder network using the corresponding cumulative loss $\sum_i L^i$
Output: Stochastic encoder network, actor network and critic network

During the adaptation step (shown in Algorithm 5.2), the parameters of all the networks will not be updated. The agent will first sample the belief z from the prior and then collected data using $\pi_\theta(a|s,z)$ given sampled z. Then the collected data will be used to update the belief, and thus, the agent can adapt to different environments. The backbone used to do the meta training is the standard state-of-the-art off-policy RL algorithm, i.e., SAC. For more details of the SAC, please refer to [170].

Algorithm 5.2: PEARL Meta Testing

Initialization:
Test task T from the distribution $f(T)$
Initialized context $c_T = \{\}$ and reply buffer $B_T = \{\}$
for $k = 1 \ldots\ldots K$ **do**
Sample belief z from inference network $q_\phi(z|c_T)$
Gather data using policy $\pi_\theta(a|s,z)$ given sampled z and add to reply buffer B_T
Update $c_T = c_T \cup B_T$
Output: The adapted actor network and critic network

5.4 Discretionary Lane Change Environment Distribution

In this section, the discretionary lane change environment distribution will be introduced. The previously introduced approaches will be used to train an agent that can adapt to that distribution of environments. As a continuity work from Chapter 4, we want to tackle the environments with a various number of trained attackers and decrease the overall crash rate by on-line adaptation. For a more realistic application, we also implement the PEARL in the distribution of environments with a broader range of driver behavior, which uses the IDM and Mobil model.

5.4.1 Environments with Attackers

In Chapter 4, we showed that the trained attacker could find failure cases that are the AV agent's fault, thus increase the crash rate. In this chapter, we want to show that by implementing MRL, the trained AV agent can adapt to different environments and thus lower the crash rate again. Therefore, as shown in Equation (5.6), we developed the distribution of environments with three variables: 1) the traffic density variable α_{den}, which is a scale of average distance between vehicles; 2) the number of total vehicles n_{car}, which can be sampled from 10 to 30; and

3) the number of attackers n_{att}, which indicate how many attackers are put around the AV, is sampled from 0 to 3. The attackers will be put around the AV randomly.

$$M_i = \{\alpha_{den},\ n_{car},\ n_{att}\},\ \ \alpha_{den} \sim U(0.5, 1.5),\ \ n_{car} \sim U\{10, 30\},\ \ n_{att} \sim U\{0,3\} \qquad (5.6)$$

Those variables are uniformly sampled and will decide the initial condition of one environment. The α_{den} can continuously sample from 0.5 to 1.5, which means at the beginning, the longitudinal distance between two cars in the same lane can sample from 9 meters to 240 meters. While the n_{car} and n_{att} can only be sampled from integers within the boundaries. After sampling these three variables, we will have a variety of environments, as shown in Figure 5.4. The reward functions for different environments keep the same as in Section 2.3.1 .

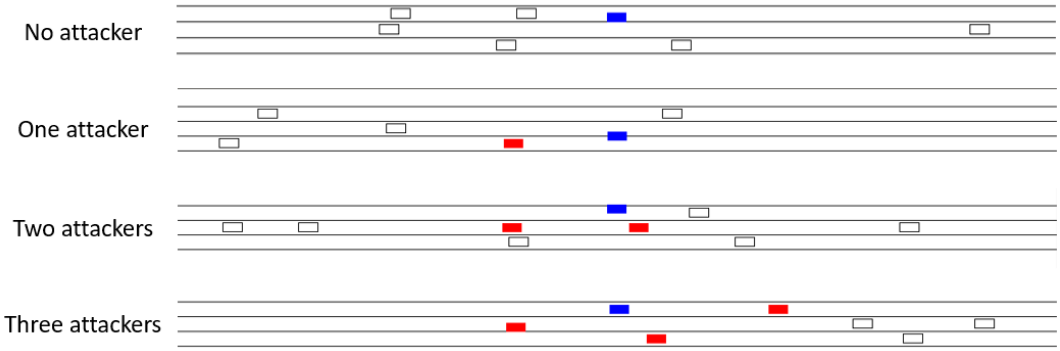


Figure 5.4 Examples of environments with different numbers of attackers (red boxes)

5.4.2 IDM-Mobil Driver Model Environments

We also implement the PEARL algorithm in another distribution of environments. In this experiment, we build the distribution of environments based on the highway-env [171] environment.

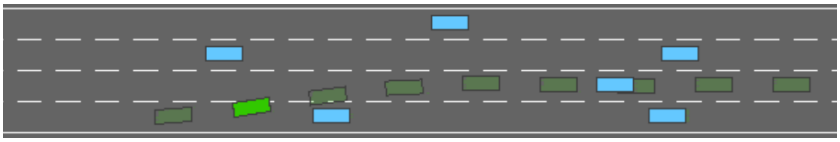


Figure 5.5 The highway-env environment [171]

The state-space $S \subseteq R^n$ of the learning agent (the green box in Figure 5.5) includes the host vehicle's lateral position y, host vehicle's longitudinal velocity v_x and the relative longitudinal position of the i^{th} surrounding vehicle Δx^i, and the relative lateral position of the i^{th} surrounding vehicle Δy^i and the relative longitudinal velocity of the i^{th} surrounding vehicle Δv_x^i. Therefore, in total, we have a continuous state space of $2 + 3 \times 6(\text{cars}) = 20$ dimensions, i.e., $S \subseteq R^{20}$. The actions of the learning agent are the *steering angle* and *acceleration*, which are both continuous. The steering angle's range is $[-\pi/4, \pi/4]$, and the acceleration's range is $[-6\ m/s^2, 6\ m/s^2]$.

In the highway-env environment, the lane change policy for the surrounding vehicles is the IDM-Mobil model, and the vehicle will change lane when:

$$\tilde{a}_c - a_c + p[(\tilde{a}_n - a_n) + (\tilde{a}_o - a_o)] > \Delta a_{th} \tag{5.7}$$

$$\tilde{a}_n > -b_{safe} \tag{5.8}$$

where the a_c is the ego vehicle's acceleration in the current lane and $\tilde{a}_c$ is the potential ego vehicle's acceleration if it changes lane. New and old successors are denoted as n and o, the corresponding a is the current acceleration and the $\tilde{a}$ is the potential if the ego vehicle changes lane. The p is the politeness factor and the Δa_{th} is the switching threshold. Therefore, the aggressiveness of the surrounding vehicle can be represented by the parameter p and Δa_{th}. And Equation (5.8) is the safety criterion guarantees that after the lane change, the deceleration of the successor in the target lane does not exceed a given safe limit b_{safe}. Since the politeness factor and the switching threshold are correlated for one kind of driver behavior, we do not sample them separately. Instead, we designed three different kinds of driver behavior: the aggressive driver, the normal driver, and the conservative driver. The corresponding parameters are listed in Table 5.1. From the table, we can see that the aggressive driver will not consider other surrounding vehicles and will change lanes with a small acceleration gain, while the conservative driver will consider other surrounding vehicles and will change lanes when there is a big acceleration gain. The normal driver is just in between.

Table 5.1 Mobil parameters for different driver behaviors

Parameters	Aggressive driver	Normal driver	Conservative driver
p	0	0.3	0.5
Δa_{th}	$0.8\,m/s^2$	$1\,m/s^2$	$1.2\,m/s^2$
b_{safe}	$2\,m/s^2$	$1\,m/s^2$	$0.5\,m/s^2$

Then one environment is decided by the following variables: the traffic density variable α_{den}, which is a scale of the average distance between vehicles; the total number of vehicles n which is the sum of the number of aggressive drivers n_{agg}, the number of normal drivers n_{nor} and the number of conservative drivers n_{con}. To sample an environment, we first uniformly sample the traffic density variable α_{den} from 0.5 to 1.5 and the total number of vehicles n from 10 to 30. Then the numbers of different driver behaviors (i.e. n_{agg}, n_{nor} and n_{con}) are sampled from the multinomial distribution $Multi(n,k)$, where n is the total number of vehicle and $k = \frac{1}{3}$. By sampling from $Multi(n,k)$, we will have $n_{agg} + n_{nor} + n_{con} = n$ and the probability of sampling from each category is the same. The reward functions [171] for different environments are the same and is composed of a velocity term and collision term:

$$R(s,a) = \alpha\left(\frac{v - v_{min}}{v_{max} - v_{min}}\right) - \beta r_{collision} \tag{5.9}$$

where v, v_{min}, and v_{max} are the current, minimum, and maximum speed of the agent, respectively, and α, β are two coefficients. For the details of the reward design, please refer to [171].

5.5 Training Setup

5.5.1 Baselines for the Meta Training and Adaptation

To show the PEARL approach's data efficiency, we compare its meta training process with the gradient-based MRL method MAML's [166] meta training process. The results are compared with the x-axis being the total collected data. And the retunes of each algorithm are averaged across five random runs. Both the hyperparameters of the PEARL and MAML are tuned carefully (PEARL: manually; MAML: by the optuna [172] package), which is an open-source hyperparameter optimization framework to automate hyperparameter search.

For the adaptation step, we compare the PEARL adaptation step with the MAML adaptation step and a fine-tune method based on the Trust Region Policy Optimization (TRPO) [173] method with safety check implemented (from Section 2.3.5). The fine-tune method will just keep updating the initial policy in a new environment. The adaptation results will be compared with the x-axis being the data collected in the new environment. We will sample 10^4 different environments and evaluate all three adaptation approaches. To evaluate the safety of the trained policy, we also calculate each trained agent's crash rate.

5.5.2 Training Hyperparameters

In this section, the hyperparameters used for PEARL are listed in Table 5.2. The hyperparameters are tuned using the Optuna [172] package. As shown in Table 5.2, the agent will be meta trained in 8 environments, and at each episode, it will be meta tested in 2 unseen environments.

Table 5.2 Implementation Hyperparameters for PEARL

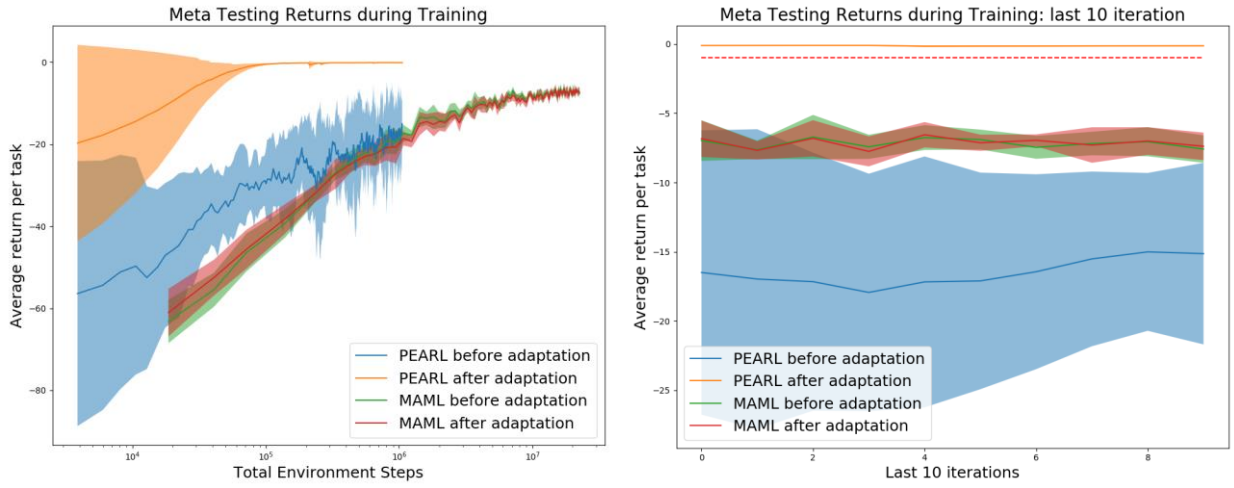
	Description	Value
n_{train}	Number of tasks used for meta training	8
n_{test}	Number of unseen tasks used for meta testing	2
n_{eval}	After the meta training, the agent will be evaluated in n_{eval} numbers of randomly sampled tasks	10^4
E	Number of episodes for each task in meta testing	2
Δt	Sampling time	0.1 sec
γ	Discount factor	0.9
n_{prior}	Number of transitions collected per task with $z \sim q_\phi(z\|c_i)$	200
K	Number of SAC iterations in each episode	1000
η	Reward scale for the backbone SAC method [170]	100
ρ	Learning rate for the SAC method [170]	3×10^{-4}
b_{meta}	Number of tasks to average the gradient across	8
b_{RL}	Number of transitions in the SAC batch [170]	256

5.6 Results

5.6.1 Training Results

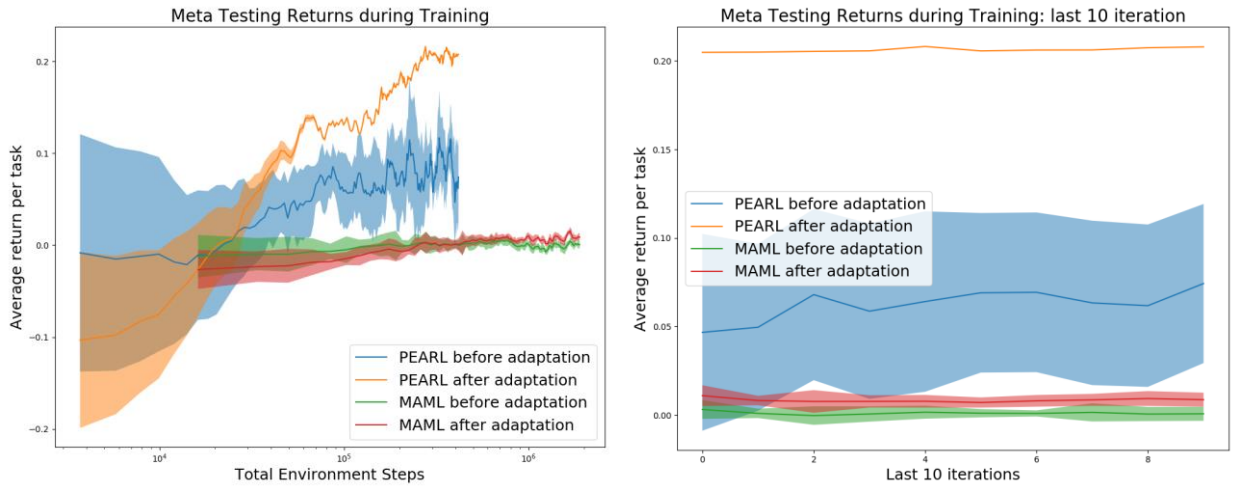
This section shows the meta testing returns of the PEARL method and MAML method during the meta training. Results for the attacker environment described in Section 5.4.1 are shown in Figure 5.6. In Figure 5.6 (a), we show the before and after adaptation of PEARL and MAML in the logarithmic axis. The x-axis is the total environment steps representing how much data they use for training. As can be seen from the figure, the PEARL method converges after collecting 10^5 data points, meanwhile the MAML converge after collecting 10^7 data points. PEARL is one hundred times more data-efficient than MAML. Moreover, if we look at the before and after adaptation curve of each approach, we can see that the agent trained by the PEARL method shows good adaptation. While for the MAML method, there is almost no adaptation.

If we zoom in on the last ten iterations of MAML and PEARL and put them together, we can have this Figure 5.6 (b). The red dashed line is the crash line. The average reward below this line indicates there are crashes in that iteration. And you can see, the MAML not only shows no adaptation, but there are also still many crashes at the end of the training. While for PEARL, we can see that there is no crash after adaptation.



(a) Total environment steps (b) Last 10 iterations

Figure 5.6 Meta Testing Average Returns during Meta Training in Attacker Environments



(a) Total environment steps (b) Last 10 iterations

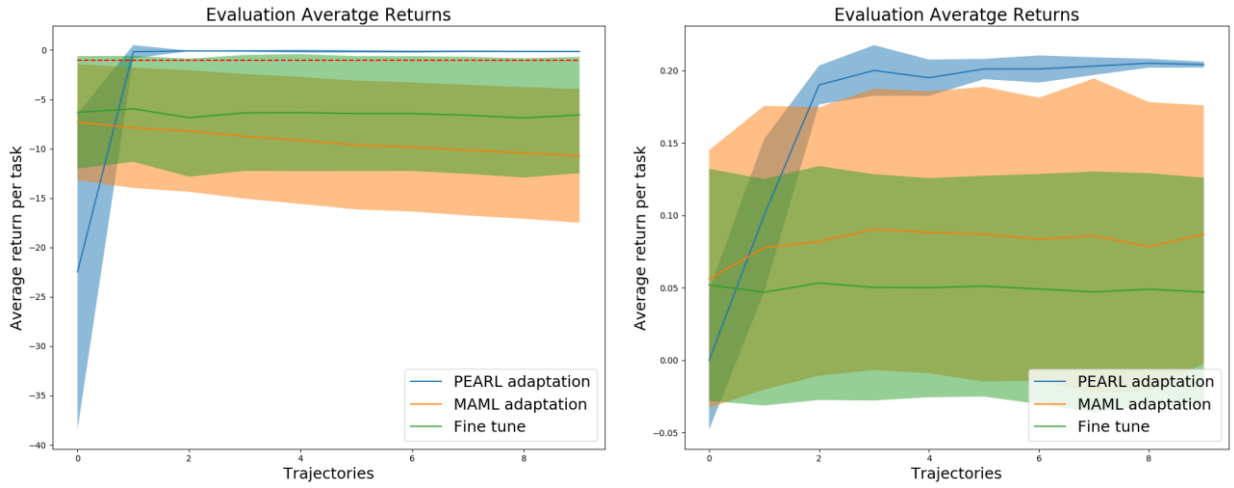
Figure 5.7 Meta Testing Average Returns during Meta Training in IDM-Mobil Environments

Results for the IDM-Mobil environment described in Section 5.4.2 are shown in Figure 5.7. In Figure 5.7 (a), we show the before and after adaptation of PEARL and MAML in the logarithmic axis, and in Figure 5.7 (b), we offer the last ten iterations of the MAML and PEAR training curve. We can have a similar conclusion that the PEARL method is much more data-efficient than the MAML method. Moreover, let us look at the before and after adaptation curve of each approach. We can see that the PEARL agent shows good adaptation that the after adaptation reward is much higher than the before adaptation reward. Since the reward design of the IDM-Mobil is different from the attacker's environment, there is no intuitive crash line. Therefore, we only summarize the crash rate in Table 5.4 in Section 5.6.2 .

5.6.2 Evaluation Results

In this section, we evaluate the trained agent with 10^4 random tasks sampled from each distribution of environments. We compared the PEARL approach with the MAML and the fine-tune approach in which we keep training the policy in a new environment. The x-axis is how much data we provide for the adaptation step after training. As you can see, after collecting two trajectories of data (400 data points), the PEARL can adapt to new environments well in both distributions of environments. However, the MAML and fine-tune methods do not show

improvement even with ten trajectories of data. This is because that the collected data in the new environment are not useful for the MAML agent and fine-tune agent to update its policies.



(a) Attackers Environment (b) IDM-Mobil Environment

Figure 5.8 Evaluation Average Returns

Next, we report the different agents' crash rates during evaluation in Table 5.3 and Table 5.4 for the attacker environments and IDM-Mobil environments, respectively. All the methods are evaluated in 10^4 random environments. On the leftmost column, we have the benchmark policy from Chapter 2. The crash rate of the trained agent in the original environment is very low. However, when we test it in random environments, the crash rate increases significantly in both setups. For the fine-tune approach, the result shows that the agent cannot adapt to new environments with limited data, so the crash rate in new environments is around the same level for both setups.

In the attacker environments, the MAML keeps getting worse and worse, give the data. This due to insufficient exploration during the adaptation. Meanwhile, the PEARL can adapt to a new environment quickly with limited data. The crash rate of the PEARL agent reaches a very small number, which can compare to the benchmark's crash rate in the original environment.

Table 5.3 Crash Rate with Different Numbers of Data in the Attacker Environments

Crash Rate	Benchmark (Safety Check)	Fine Tune (Safety Check)	MAML (No Safety Check)	PEARL (No Safety Check)
In orig. task	$\sim 10^{-3}\%$	$\sim 10^{-3}\%$	-	-
Before adapting	17.8%	13.2%	19.4%	59.1%
1 trajectory		13.4%	22.5%	7.3%
2 trajectories		14.1%	24.9%	0.099%
3 trajectories		13.3%	27.3%	0.077%
5 trajectories		13.7%	31.4%	0.015%
10 trajectories		13.8%	36.9%	**0.0062%**

Table 5.4 Crash Rate with Different Numbers of Data in the IDM-Mobil Environments

Crash Rate	Benchmark	Fine Tune	MAML	PEARL
In orig. task	~4%	~4%	-	-
Before adapting	50.3%	52.6%	50.4%	60.3%
1 trajectory		51.9%	32.6%	26.7%
2 trajectories		49.5%	36.7%	18.1%
3 trajectories		49.8%	34.5%	12.7%
5 trajectories		48.2%	32.2%	10.5%
10 trajectories		47.6%	31.8%	**5.2%**

In IDM-Mobil environments, the MAML agent has better crash rates with more and more given data. However, the improvement still not significant enough compared to the PEARL agent. As can be seen from Table 5.4, the PEARL can adapt to a new environment quickly with limited data. The crash rate of the PEARL is comparable to the benchmark's crash rate in the original environment. Since in the IDM-Mobil environments, there is no short-horizon safety check, the benchmark crash rate is higher than the attacker environment. Moreover, in the IDM-Mobil environments, the agent controls the steering angle and the acceleration directly without any robust lower level controller. This causes a higher crash rate compared to the attacker environment. The crash rate results show that the PEARL trained agent can achieve the benchmark level crash rate with only ten trajectories of data in both setups.

5.7 Summary

In this chapter, we showed that it is necessary to solve the multi-MDPs problem in designing the DLC policy. At a different time of the day or in different weather conditions, drivers can behave differently. And from Chapter 4 we know that the DNN-based policy will fail with perturbations.

To solve the multi-MDPs problem and design a robust DLC policy, we can try to encode the knowledge from previously experienced environments and utilize it in some unseen environments. This chapter implements the state-of-the-art method meta reinforcement learning (MRL) method PEARL to encode the knowledge into a stochastic encoder.

Two distributions of environments are designed in this chapter, i.e., the environments with attackers and the IDM-Mobil environments in Section 5.4 . From the results, we can see that, in both distributions of environments, the trained policy can adapt to unseen environments with limited data. While the baseline fine-tune method cannot adapt to unseen environments with the same amount of data. Moreover, we also calculate the crash rate of the trained agent. As shown in the results, the crash rate of the trained agent decreases dramatically after the adaptation. With only ten trajectories, the crash rate can drop significantly to the crash rate of the benchmark policy trained in the original environment. From the results, we can see that the MRL method has huge potentials in solving the AV decision-making problem and can provide a robust policy concerning the MDP transition probability.

Chapter 6 Conclusions and Future Works

6.1 Conclusions

To develop an automated vehicle with higher automation levels, designing the discretionary lane change (DLC) policy is critical. However, the DLC policy is hard to design using the conventional method since it needs to consider complicated environment information and different conditions. Therefore, in Chapter 2, we first proposed a novel RL method that uses the model-based exploration method via intrinsic reward to synthesize the DLC policy. In particular, an environment transition model is trained as a notion of an agent's surprise about its experiences guide the exploration. The agent thus can explore the state space thoroughly and result in an optimal global policy. The experiments we conduct show that the model-based exploration method we proposed leads to a faster convergence solution and designed a DLC policy that can travel more efficiently. The model-based exploration method we developed offers both theoretical and practical advantages in solving the DLC problem.

After designing a decision-making system, the policy must be evaluated thoroughly before its release and deployment. In Chapter 3, we evaluated the policy designed in Chapter 2. To assess the policy efficiently in a high dimensional state space, we implemented the Subset Simulation (SS) as an adaptive sampling method for accelerated evaluation. We demonstrated the ability of SS to accelerate the evaluation in Chapter 3. The SS method is proved to have better-accelerating performance than the Importance Sampling (IS) method, and it can evaluate the system in the environment with a high dimension while the IS method cannot.

The limitation of the SS method is that the "danger regions" are searched as the test procedure unfolds. If the environmental statistics change, the crash rate cannot be estimated accurately. Therefore, we developed a novel evaluation method in Chapter 4 that does not need environmental statistics. In Chapter 4, we first design an attacker under the two-player Markov game framework to challenge the AV. The attacker is trained to generate socially acceptable attacks by well designing the reward function that considers the accident's responsibility. The attacker's objective is to lure the AV to end up in AV-responsible crashes. After the attacker's

policy is trained to converge, the AV is evaluated in the environment with one attacker. The evaluation results show that the attacker can lure the AV into many AV-responsible crashes, and the average crash rate increases 50 times.

Introducing attackers that can generate socially acceptable attacks makes the behavior of the surrounding vehicles more diverse. The trained policy from Chapter 2 failed in such varied environments. This problem can be viewed as designing a robust policy with respect to multiple MDPs. In Chapter 5, we solved this multi-MDPs problem. In detail, we encode the knowledge from previously experienced environments and utilize it in some unseen environments. We implemented the state-of-the-art method meta reinforcement learning (MRL) method PEARL to encode the knowledge into a stochastic encoder. Two distributions of environments are designed in this chapter, i.e., the environments with attackers and IDM-Mobil environments. From the results, we can see that the trained policy can adapt to unseen environments with only two trajectories of data in both distributions of environments. Moreover, the crash rate of the trained agent decreases dramatically after the adaptation to the crash rate of the benchmark policy trained in the original environment. From the results, we can see that the MRL method has huge potentials in solving the AV decision-making problem.

This dissertation discussed the design procedure (as shown in Figure 6.1) of the DLC policy, from the designing stage to the evaluating stage and, finally, using the evaluation results to further improve the policy. The developed evaluation methods and MRL method have great potentials to solve for robust policies concerning a wide range of conditions in different applications.

Figure 6.1 Procedure for developing an AV's decision-making system

6.2 Future Works

The developed designing and evaluation methods proposed in this dissertation can be used in solving more decision-making problems. While we successfully applied them in solving the DLC problem, more can be done to extend the methods to other scenarios. A few research directions that can be exploited in future research are discussed as follows.

6.2.1 Designing Robust Decision-making Systems of Other AV Scenarios

New training approaches need to be developed for more application, considering different weather conditions, road topology, and a more comprehensive range of human behaviors. More works are also required to design a decision-making system for the AV in other scenarios, i.e., the highway merging and exiting problem, roundabout entering problem, etc.

Moreover, to implement the learned policy in the real-world application, it is necessary to calibrate the distribution of the simulators. This requires collecting vast amounts of real-world data to ensure that the built simulators cover the range of behavior and conditions in the real world.

Finally, to implement the learned policy in an AV system in practice, we also need to consider integrating the decision-making system with other systems. Challenges can come from sensing, perception/detection, motion planning, and control systems. These integration problems should also be studied carefully to get a safe and robust AV product.

6.2.2 Online Monitoring Environment Changes

The MRL method we implement can only start to adapt to the new environment when noticed manually. However, the fully automated vehicle needs to decide when to adapt by itself. Therefore, online MRL methods also need to be developed. Online MRL considers a setting where either the agent monitors the environment changes and adapts when necessary, or just keeps adapting. There are some potential methods [174], but more works are needed to implement them to the AV applications.

6.2.3 Extrapolate rather than Interpolate

One of the assumptions under the MRL is that the training environments and the testing environments are sampled from the same distribution. This means the adaptation step is basically running some "interpolation." Current MRL approaches are either not able to extrapolate well or can do so at the expense of requiring vast amounts of online collecting data. Solving the extrapolation problem is an open question. It requires the implementation of some structure that represents common knowledge or reasoning inside the policy. This is the next big step towards human-level intelligence.

Bibliography

[1] Society of Automotive Engineers, “Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems,” 2014.

[2] National Highway Traffic Safety Administration, “Preliminary Statement of Policy Concerning Automated Vehicles,” 2013.

[3] Germany Federal Highway Research Institute, “Legal consequences of an increase in vehicle automation,” 2013. doi: 1100.5409013.01.

[4] Wikipedia, “Self-driving car,” *Wikipedia Foundation, Inc.* 2020, [Online]. Available: https://en.wikipedia.org/wiki/Self-driving_car.

[5] D. Zhao, “Accelerated Evaluation of Automated Vehicles,” 2016.

[6] Teslapedia, “Release Notes: 2019.40.50.1,” *Teslascope*. 2019, [Online]. Available: https://teslascope.com/teslapedia/software/2019.40.50.1.

[7] P. Bigelow, “Why Level 3 automated technology has failed to take hold,” *Automotive News*, 2019.

[8] R. Bishop, “OEDR: The Key Differentiator Between SAE Level 2 And Level 3 Automated Driving,” *Forbes*, 2019.

[9] P. Lyon, “Nissan Reveals Revolutionary Hands-Off Self-Driving Tech,” *Forbes*, 2019.

[10] J. Stewart, “All the Startups and Companies Working on Self-Driving Cars,” *WIRED*, 2017. https://www.wired.com/2017/05/mapped-top-263-companies-racing-toward-autonomous-cars/ (accessed Aug. 02, 2020).

[11] Wikipedia, “List of self-driving car fatalities,” *Wikipedia Foundation, Inc.* 2019, [Online]. Available: https://en.wikipedia.org/wiki/List_of_self-driving_car_fatalities.

[12] C. Badue *et al.*, “Self-Driving Cars: A Survey,” 2019, [Online]. Available: http://arxiv.org/abs/1901.04407.

[13] M. Montemerlo *et al.*, “Junior: The Stanford entry in the urban challenge,” *J. F. Robot.*, vol. 25, no. 9, pp. 569–597, Sep. 2008, doi: 10.1002/rob.20258.

[14] A. Kesting, M. Treiber, and D. Helbing, “General Lane-Changing Model MOBIL for Car-Following Models,” *Transp. Res. Rec. J. Transp. Res. Board*, vol. 1999, no. 1, pp. 86–94, 2007, doi: 10.3141/1999-10.

[15] M. Treiber, A. Hennecke, and D. Helbing, “Congested traffic states in empirical observations and microscopic simulations,” *Phys. Rev. E - Stat. Physics, Plasmas, Fluids, Relat. Interdiscip. Top.*, vol. 62, no. 2, pp. 1805–1824, 2000, doi: 10.1103/PhysRevE.62.1805.

[16] L. Zhao, R. Ichise, T. Yoshikawa, T. Naito, T. Kakinami, and Y. Sasaki, “Ontology-based decision making on uncontrolled intersections and narrow roads,” in *IEEE Intelligent Vehicles Symposium, Proceedings*, 2015, vol. 2015-Augus, pp. 83–88, doi: 10.1109/IVS.2015.7225667.

[17] L. Zhao, R. Ichise, Z. Liu, S. Mita, and Y. Sasaki, “Ontology-based driving decision making: A feasibility study at uncontrolled intersections,” *IEICE Trans. Inf. Syst.*, vol. E100D, no. 7, pp. 1425–1439, 2017, doi: 10.1587/transinf.2016EDP7337.

[18] J. Nilsson and J. Sjoberg, “Strategic decision making for automated driving on two-lane, one way roads using model predictive control,” in *IEEE Intelligent Vehicles Symposium, Proceedings*, 2013, pp. 1253–1258, doi: 10.1109/IVS.2013.6629638.

[19] M. Bojarski *et al.*, “End to End Learning for Self-Driving Cars,” 2016, Accessed: Apr. 23, 2020. [Online]. Available: http://arxiv.org/abs/1604.07316.

[20] F. Codevilla, M. Miiller, A. Lopez, V. Koltun, and A. Dosovitskiy, “End-to-End Driving Via Conditional Imitation Learning,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2018, pp. 4693–4700, doi: 10.1109/ICRA.2018.8460487.

[21] Y. Guan, S. E. Li, J. Duan, W. Wang, and B. Cheng, “Markov probabilistic decision making of self-driving cars in highway with random traffic flow: a simulation study,” *J. Intell. Connect. Veh.*, vol. 1, no. 2, pp. 77–84, Jun. 2018, doi: 10.1108/jicv-01-2018-0003.

[22] Z. Cao *et al.*, “Highway Exiting Planner for Automated Vehicles Using Reinforcement Learning,” *IEEE Trans. Intell. Transp. Syst.*, 2020.

[23] M. Mukadam, A. Cosgun, A. Nakhaei, and K. Fujimura, “Tactical Decision Making for Lane Changing with Deep Reinforcement Learning,” *Neural Inf. Process. Syst.*, no. Nips, pp. 1–10, 2017, Accessed: Apr. 03, 2020. [Online]. Available: https://openreview.net/forum?id=B1G6uM0WG.

[24] S. Nageshrao, E. Tseng, and D. Filev, “Autonomous Highway Driving using Deep Reinforcement Learning,” 2019, [Online]. Available: http://arxiv.org/abs/1904.00035.

[25] L. Wen, J. Duan, S. E. Li, S. Xu, and H. Peng, “Safe Reinforcement Learning for Autonomous Vehicles through Parallel Constrained Policy Optimization,” no. 2012, 2020, [Online]. Available: http://arxiv.org/abs/2003.01303.

[26] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An Open Urban Driving Simulator,” 2017, Accessed: Apr. 23, 2020. [Online]. Available: http://arxiv.org/abs/1711.03938.

[27] C. You, J. Lu, D. Filev, and P. Tsiotras, “Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning,” *Rob. Auton. Syst.*, vol. 114, pp. 1–18, Apr. 2019, doi: 10.1016/j.robot.2019.01.003.

[28] S. Kuutti, R. Bowden, H. Joshi, R. De Temple, and S. Fallah, “End-to-end Reinforcement Learning for Autonomous Longitudinal Control Using Advantage Actor Critic with Temporal Context,” in *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019*, 2019, pp. 2456–2462, doi: 10.1109/ITSC.2019.8917387.

[29] M. Jaritz, R. De Charette, M. Toromanoff, E. Perot, and F. Nashashibi, “End-to-End Race Driving with Deep Reinforcement Learning,” in *Proceedings - IEEE International*

Conference on Robotics and Automation, 2018, pp. 2070–2075, doi: 10.1109/ICRA.2018.8460934.

[30] National Highway Traffic Safety Administration, "Federal Motor Vehicle Safety Standards," 1998. [Online]. Available: https://www.nhtsa.gov/laws-regulations/fmvss.

[31] National Highway Traffic Safety Administration, "New Car Assessment Program," 2006.

[32] The World Forum for Harmonization of Vehicle Regulations, "United Nations Economic Commission for Europe Regulations," 1998.

[33] The China Automotive Technology and Research Center, "C-NCAP Management Regulation," pp. 1–222, 2018, [Online]. Available: http://www.c-ncap.org.cn/c-ncap_en/ep/2012english.pdf.

[34] W. H. Ma and H. Peng, "Worst-case evaluation methods for vehicle control systems," in *American Society of Mechanical Engineers, Dynamic Systems and Control Division (Publication) DSC*, 1996, vol. 58, pp. 83–90, Accessed: Feb. 12, 2020. [Online]. Available: https://deepblue.lib.umich.edu/handle/2027.42/131268.

[35] W. H. Ma and H. Peng, "A worst-case evaluation method for dynamic systems," *J. Dyn. Syst. Meas. Control. Trans. ASME*, vol. 121, no. 2, pp. 191–199, 1999, doi: 10.1115/1.2802454.

[36] Y. Kou, "Development and Evaluation of Integrated Chassis Control Systems," 2010.

[37] M. L. Moore, "Assurance and control of vehicle emission testing," 1973, doi: 10.4271/730534.

[38] W. G. Najm *et al.*, "Description of Light-Vehicle Pre-Crash Scenarios for Safety Applications Based On Vehicle-to-Vehicle Communications," 2013. Accessed: Feb. 13, 2020. [Online]. Available: https://rosap.ntl.bts.gov/view/dot/9980.

[39] W. G. Najm, J. D. Smith, and M. Yanagisawa, "Pre-Crash Scenario Typology for Crash Avoidance Research," *Security*, no. April, p. 128, 2007, Accessed: Feb. 13, 2020. [Online]. Available: https://rosap.ntl.bts.gov/view/dot/6281.

[40] W. G. Najm and J. D. Smith, "Development of Crash Imminent Test Scenarios for Integrated Vehicle-Based Safety Systems," no. April, p. 57p, 2007, Accessed: Feb. 13, 2020. [Online]. Available: https://rosap.ntl.bts.gov/view/dot/8883.

[41] W. G. Najm, S. Toma, and J. Brewer, "Depiction of priority light-vehicle pre-crash scenarios for safety applications based on vehicle-to-vehicle communications," 2013. Accessed: Feb. 13, 2020. [Online]. Available: https://rosap.ntl.bts.gov/view/dot/9887.

[42] A. Y. Ungoren and H. Peng, "Evaluation of vehicle dynamic control for rollover prevention," *Int. J. Automot. Technol.*, vol. 5, no. 2, pp. 115–122, 2004, Accessed: Feb. 13, 2020. [Online]. Available: https://pdfs.semanticscholar.org/357b/5b978b6cb732d8601d442a90fbc09864a0ea.pdf.

[43] Euro NCAP, "Adult Occupant Protection." https://www.euroncap.com/en/vehicle-safety/the-ratings-explained/adult-occupant-protection/.

[44] M. Althoff, "Reachability Analysis and its Application to the Safety Assessment of Autonomous Cars," 2010.

[45] M. Althoff and J. M. Dolan, “Online verification of automated road vehicles using reachability analysis,” *IEEE Trans. Robot.*, vol. 30, no. 4, pp. 903–918, 2014, doi: 10.1109/TRO.2014.2312453.

[46] M. Althoff, D. Althoff, D. Wollherr, and M. Buss, “Safety verification of autonomous vehicles for coordinated evasive maneuvers,” in *IEEE Intelligent Vehicles Symposium, Proceedings*, 2010, pp. 1078–1083, doi: 10.1109/IVS.2010.5548121.

[47] M. Althoff and S. Lutz, “Automatic Generation of Safety-Critical Test Scenarios for Collision Avoidance of Road Vehicles,” *IEEE Intell. Veh. Symp. Proc.*, vol. 2018-June, no. Iv, pp. 1326–1333, 2018, doi: 10.1109/IVS.2018.8500374.

[48] E. Clarke, O. Grumberg, D. Kroening, D. Peled, and H. Veith, *Model Checking*. MIT press, 2018.

[49] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, “Model checking and the state explosion problem,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2012, vol. 7682 LNCS, pp. 1–30, doi: 10.1007/978-3-642-35746-6_1.

[50] C. E. Tuncali, G. Fainekos, B. Amor, J. Kapinski, and A. Shrivastava, “Search-based Test Generation for Automated Driving Systems: From Perception to Control Logic,” no. May, 2019.

[51] Waymo, “Waymo open dataset,” 2020. https://waymo.com/open/.

[52] lyft, “Lyft Level 5 Dataset,” 2020. https://level5.lyft.com/dataset/.

[53] Aptiv, “NuScenes by Aptiv,” 2020. https://www.nuscenes.org/.

[54] A. Patil, S. Malla, H. Gang, and Y. T. Chen, “The H3D dataset for full-surround 3D multi-object detection and tracking in crowded urban scenes,” in *Proceedings - IEEE International Conference on Robotics and Automation*, Mar. 2019, vol. 2019-May, pp. 9552–9557, doi: 10.1109/ICRA.2019.8793925.

[55] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The KITTI dataset,” *Int. J. Rob. Res.*, vol. 32, no. 11, pp. 1231–1237, Sep. 2013, doi: 10.1177/0278364913491297.

[56] FESTA-Consortium, “FESTA Handbook Version 2 Deliverable T6. 4 of the Field opErational teSt supporT Action,” 2008.

[57] M. Ljung Aust, “Evaluation Process for Active Safety Functions: Addressing Key Challenges in Functional, Formative Evaluation of Advanced Driver Assistance Systems,” 2012, Accessed: Feb. 14, 2020. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Improving+the+Evaluation+Process+for+Active+Safety+Functions+Addressing+Key+Challenges+in+Functional+Formative+Evaluation+of+Advanced+Driver+Assistance+Systems#1.

[58] Waymo, “On the road to Fully Self-driving - Waymo Safety Report,” 2018. doi: 10.1016/B978-0-08-037539-7.50012-0.

[59] Wikipedia, “Waymo,” *Wikipedia Foundation, Inc.* 2020, [Online]. Available: https://en.wikipedia.org/wiki/Waymo.

[60] UMTRI, “Safety Pilot Model Deployment.” http://safetypilot.umtri.umich.edu/.

[61] D. Bezzina and J. Sayer, “Safety pilot model deployment: Test conductor team report,” 2015.

[62] Booz, Allen, and Hamilton, “Safety Pilot Model Deployment – One Day Sample Data Environment,” 2015.

[63] J. Sayer *et al.*, “Integrated Vehicle-Based Safety Systems Field Operational Test Final Program Report,” *Transp. Res.*, no. June, 2011, Accessed: Feb. 15, 2020. [Online]. Available: www.ntis.gov.

[64] M. Akamatsu, P. Green, and K. Bengler, “Automotive technology and human factors research: Past, present, and future,” *International Journal of Vehicular Technology*, vol. 2013. 2013, doi: 10.1155/2013/526180.

[65] J. Golson, “Google will pay Arizona drivers $20 per hour to test self-driving cars,” *The Verge*, 2016. https://www.theverge.com/2016/5/12/11668548/google-self-driving-arizona-hiring-operator.

[66] N. Kalra and S. M. Paddock, “Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?,” *Transp. Res. Part A Policy Pract.*, vol. 94, pp. 182–193, 2016, doi: 10.1016/j.tra.2016.09.010.

[67] H. H. Yang and H. Peng, “Development and evaluation of collision warning/collision avoidance algorithms using an errable driver model,” in *Vehicle System Dynamics*, 2010, vol. 48, no. SUPPL. 1, pp. 525–535, doi: 10.1080/00423114.2010.515745.

[68] R. S. Jurecki and T. L. Stańczyk, “Driver reaction time to lateral entering pedestrian in a simulated crash traffic situation,” *Transp. Res. Part F Traffic Psychol. Behav.*, vol. 27, no. PA, pp. 22–36, 2014, doi: 10.1016/j.trf.2014.08.006.

[69] Di. Zhao, X. Huang, H. Peng, H. Lam, and D. J. Leblanc, “Accelerated Evaluation of Automated Vehicles in Car-Following Maneuvers,” *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 3, pp. 733–744, 2018, doi: 10.1109/TITS.2017.2701846.

[70] D. Zhao *et al.*, “Accelerated Evaluation of Automated Vehicles Safety in Lane-Change Scenarios Based on Importance Sampling Techniques,” *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 3, pp. 595–607, 2017, doi: 10.1109/TITS.2016.2582208.

[71] Z. Huang, H. Lam, D. J. Leblanc, and D. Zhao, “Accelerated Evaluation of Automated Vehicles Using Piecewise Mixture Models,” *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 9, pp. 2845–2855, 2018, doi: 10.1109/TITS.2017.2766172.

[72] M. O’Kelly, J. Duchi, A. Sinha, H. Namkoong, and R. Tedrake, “Scalable end-to-end autonomous vehicle testing via rare-event simulation,” in *Advances in Neural Information Processing Systems*, 2018, vol. 2018-Decem, pp. 9827–9838, Accessed: Feb. 18, 2020. [Online]. Available: http://papers.nips.cc/paper/8189-scalable-end-to-end-autonomous-vehicle-testing-via-rare-event-simulation.

[73] S. K. Au and Y. Wang, *Engineering Risk Assessment with Subset Simulation*, vol. 9781118398. 2014.

[74] L. MARGOLIN, “On the Convergence of the Cross-Entropy Method,” *Ann. Oper. Res.*, p.

14, 2005, doi: 10.2307/2307868.

[75] S. Feng, Y. Feng, C. Xu, Y. Zhang, and H. X. Liu, “Testing Scenario Library Generation for Connected and Automated Vehicles, Part I: Methodology,” *Intell. Transp. Syst. IEEE Trans.*, May 2019.

[76] S. Feng, Y. Feng, H. Sun, S. Bao, Y. Zhang, and H. X. Liu, “Testing Scenario Library Generation for Connected and Automated Vehicles, Part II: Case Studies,” *Intell. Transp. Syst. IEEE Trans.*, May 2019, Accessed: Feb. 19, 2020. [Online]. Available: http://arxiv.org/abs/1905.03428.

[77] A. Talebpour, H. S. Mahmassani, and S. H. Hamdar, “Modeling Lane-Changing Behavior in a Connected Environment: A Game Theory Approach,” *Transp. Res. Procedia*, vol. 7, pp. 420–440, 2015, doi: 10.1016/j.trpro.2015.06.022.

[78] D. Silver *et al.*, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, 2016, doi: 10.1038/nature16961.

[79] C.-J. Hoel, K. Wolff, and L. Laine, “Automated Speed and Lane Change Decision Making using Deep Reinforcement Learning,” 2018, [Online]. Available: http://arxiv.org/abs/1803.10056.

[80] I.-A. Hosu and T. Rebedea, “Playing Atari Games with Deep Reinforcement Learning and Human Checkpoint Replay,” Jul. 2016, Accessed: Jul. 30, 2020. [Online]. Available: http://arxiv.org/abs/1607.05077.

[81] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, “Dueling Network Architectures for Deep Reinforcement Learning,” no. 9, 2015, doi: 10.1109/MCOM.2016.7378425.

[82] H. van Hasselt, A. Guez, and D. Silver, “Deep Reinforcement Learning With Double Q-Learning,” pp. 2094–2100, 2016, [Online]. Available: http://arxiv.org/abs/1606.04615.

[83] D. Silver, “Lecture 4: Model-Free Prediction.” https://www.davidsilver.uk/wp-content/uploads/2020/03/MC-TD.pdf (accessed Jul. 30, 2020).

[84] OpenAI, “Vanilla Policy Gradient — Spinning Up documentation.” https://spinningup.openai.com/en/latest/algorithms/vpg.html (accessed Jul. 30, 2020).

[85] Chris Yoon, “Understanding Actor Critic Methods – Towards Data Science.” https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f (accessed Jul. 30, 2020).

[86] T. Mitchell, “0703 Deep Reinforcement Learning Policy Gradient Methods-Part 2,” 2018.

[87] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, “High-dimensional continuous control using generalized advantage estimation,” 2016, Accessed: Jul. 30, 2020. [Online]. Available: https://sites.google.com/site/gaepapersupp.

[88] J. Wang, Q. Zhang, D. Zhao, and Y. Chen, “Lane Change Decision-making through Deep Reinforcement Learning with Rule-based Constraints,” in *Proceedings of the International Joint Conference on Neural Networks*, 2019, vol. 2019-July, doi: 10.1109/IJCNN.2019.8852110.

[89] K. Murphy, “Dynamic Bayesian Networks: Representation, Inference and Learning,”

Univ. California, Berkeley, p. 223, 2002, doi: 10.1.1.129.7714.

[90] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, 1996, doi: 10.1613/jair.301.

[91] S. B. Thrun, "Efficient Exploration In Reinforcement Learning," *Science (80-.).*, no. January, pp. 1–44, 1992, doi: 10.1109/IJCNN.2001.939497.

[92] J. Z. Kolter and A. Y. Ng, "Near-bayesian exploration in polynomial time," in *ACM International Conference Proceeding Series*, 2009, vol. 382, doi: 10.1145/1553374.1553441.

[93] R. I. Brafman and M. Tennenholtz, "R-MAX - A general polynomial time algorithm for near-optimal reinforcement learning," in *IJCAI International Joint Conference on Artificial Intelligence*, 2001, vol. 3, pp. 953–958, Accessed: Jun. 29, 2020. [Online]. Available: http://www.jmlr.org/papers/v3/brafman02a.html.

[94] J. Achiam and S. Sastry, "Surprise-Based Intrinsic Motivation for Deep Reinforcement Learning," pp. 1–13, 2017, doi: 10.1051/0004-6361/201527329.

[95] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-Driven Exploration by Self-Supervised Prediction," *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Work.*, 2017, doi: 10.1109/CVPRW.2017.70.

[96] G. Vezzani, A. Gupta, L. Natale, and P. Abbeel, "Learning latent state representation for speeding up exploration," 2019, [Online]. Available: http://arxiv.org/abs/1905.12621.

[97] W. Schakel, V. Knoop, and B. van Arem, "Integrated Lane Change Model with Relaxation and Synchronization," *Transp. Res. Rec. J. Transp. Res. Board*, vol. 2316, no. 2316, pp. 47–57, 2012, doi: 10.3141/2316-06.

[98] C. Doersch, "Tutorial on Variational Autoencoders," pp. 1–23, 2016.

[99] K. Sohn, H. Lee, and X. Yan, "Learning Structured Output Representation using Deep Conditional Generative Models," *Adv. Neural Inf. Process. Syst.*, pp. 3483–3491, 2015, [Online]. Available: http://papers.nips.cc/paper/5775-learning-structured-output-representation-using-deep-conditional-generative-models.

[100] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," no. Ml, pp. 1–14, 2013, [Online]. Available: http://arxiv.org/abs/1312.6114.

[101] K. Zuev, "Subset Simulation Method for Rare Event Estimation: An Introduction," 2015, doi: 10.1007/978-3-642-36197-5_165-1.

[102] A. Lagnoux, "Rare Event Simulation," *Probab. Eng. Informational Sci.*, vol. 20, no. 01, pp. 45–66, 2005, doi: 10.1017/S0269964806060025.

[103] R. Rubinstein and D. Kroese, *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. 2013.

[104] R. Rubinstein and B. Melamed, *Modern simulation and modelling*. 1998.

[105] S. K. Au and J. L. Beck, "Important sampling in high dimensions," *Struct. Saf.*, vol. 25, no. 2, pp. 139–163, 2003, doi: 10.1016/S0167-4730(02)00047-4.

[106] L. S. Katafygiotis and K. M. Zuev, "Geometric insight into the challenges of solving high-

dimensional reliability problems," *Probabilistic Eng. Mech.*, vol. 23, no. 2–3, pp. 208–218, 2008, doi: 10.1016/j.probengmech.2007.12.026.

[107] J. S. Liu, *Monte Carlo Strategies in Scientific Computing*. 2004.

[108] S. K. Au and J. L. Beck, "Estimation of small failure probabilities in high dimensions by subset simulation," *Probabilistic Eng. Mech.*, vol. 16, no. 4, pp. 263–277, 2001, doi: 10.1016/S0266-8920(01)00019-4.

[109] I. Papaioannou, W. Betz, K. Zwirglmaier, and D. Straub, "MCMC algorithms for Subset Simulation," *Probabilistic Eng. Mech.*, vol. 41, pp. 89–103, 2015, doi: 10.1016/j.probengmech.2015.06.006.

[110] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *J. Chem. Phys.*, vol. 21, no. 6, pp. 1087–1092, 1953, doi: 10.1063/1.1699114.

[111] K. M. Zuev, J. L. Beck, S. K. Au, and L. S. Katafygiotis, "Bayesian post-processor and other enhancements of Subset Simulation for estimating failure probabilities in high dimensions," *Comput. Struct.*, vol. 92–93, pp. 283–296, 2012, doi: 10.1016/j.compstruc.2011.10.017.

[112] L. Barr and W. Najm, "Crash Problem Characteristics for the Intelligent Vehicle Initiative," *Natl. Res. Counc. (U.S.). Transp. Res. Board. Meet. (80th 2001 Washington, D.C.). Prepr. CD-ROM*, no. 01, p. 30 p., 2001, Accessed: Mar. 02, 2020. [Online]. Available: https://trid.trb.org/view/675725.

[113] G. M. Fitch, S. E. Lee, S. Klauer, J. Hankey, J. Sudweeks, and T. Dingus, "Analysis of Lane-Change Crashes and Near-Crashes," *Final Rep. DOT HS 811 147, US Dep. Transp. Natl. Highw. Traffic Saf. Adm.*, no. June, pp. 1–88, 2009, doi: DOT HS 811 147.

[114] D. S. Gurupackiam and S. Jones, "Characterization of Arterial Traffic Congestion Through Analysis of Operational Parameters (Gap Acceptance and Lane Changing)," *Report*, no. 07112, 2010, [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Characterization+of+Arterial+Traffic+Congestion+Through+Analysis+of+Operational+Parameters#1%5Cnhttp://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Characterization+of+Arterial+Traffi.

[115] Z. Zheng, "Recent developments and research needs in modeling lane changing," *Transp. Res. Part B Methodol.*, vol. 60, pp. 16–32, 2014, doi: 10.1016/j.trb.2013.11.009.

[116] D. Zhao, H. Peng, K. Nobukawa, S. Bao, D. J. LeBlanc, and C. S. Pan, "Analysis of mandatory and discretionary lane change behaviors for heavy trucks," *arxiv.org*, 2017, Accessed: Mar. 02, 2020. [Online]. Available: https://arxiv.org/abs/1707.09411.

[117] T. A. Dingus *et al.*, "The 100-Car Naturalistic driving Study Phase II – Results of the 100-Car Field Experiment," 2006. doi: DOT HS 810 593.

[118] R. N. Jazar, *Vehicle dynamics: Theory and Application*, vol. 24, no. 4. 2014.

[119] Wikipedia, "Correlation and dependence," *Wikipedia Foundation, Inc.* 2020, [Online]. Available: https://en.wikipedia.org/wiki/Correlation_and_dependence.

[120] V. Ramanujam, “Lane Changing Models for Arterial Traffic,” *Environ. Eng.*, no. 2005, 2007, [Online]. Available: http://hdl.handle.net/1721.1/39286.

[121] P. Angkititrakul, C. Miyajima, and K. Takeda, “Stochastic Mixture Modeling of Driving Behavior During Car Following,” *J. Inf. Commun. Converg. Eng.*, vol. 11, no. 2, pp. 95–102, 2013, doi: 10.6109/jicce.2013.11.2.095.

[122] P. Angkititrakul, C. Miyajima, and K. Takeda, “Modeling and adaptation of stochastic driver-behavior model with application to car following,” *IEEE Intell. Veh. Symp. Proc.*, no. Iv, pp. 814–819, 2011, doi: 10.1109/IVS.2011.5940464.

[123] X. Huang, S. Zhang, and H. Peng, “Developing Robot Driver Etiquette Based on Naturalistic Human Driving Behavior,” *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 4, pp. 1–11, Apr. 2019, doi: 10.1109/tits.2019.2913102.

[124] D. Reynolds, “Gaussian Mixture Models,” *Encycl. Biometrics*, no. 2, pp. 659–663, 2009, doi: 10.1007/978-0-387-73003-5_196.

[125] Z. Ju and H. Liu, “Fuzzy Gaussian mixture models,” *Pattern Recognit.*, vol. 45, no. 3, pp. 1146–1158, 2012, doi: 10.1016/j.patcog.2011.08.028.

[126] T. M. Nguyen, Q. M. Jonathan Wu, and H. Zhang, “Bounded generalized Gaussian mixture model,” *Pattern Recognit.*, vol. 47, no. 9, pp. 3132–3142, 2014, doi: 10.1016/j.patcog.2014.03.030.

[127] G. Lee and C. Scott, “EM algorithms for multivariate Gaussian mixture models with truncated and censored data,” *Comput. Stat. Data Anal.*, vol. 56, no. 9, pp. 2816–2829, 2012, doi: 10.1016/j.csda.2012.03.003.

[128] K. P. Burnham and D. R. Anderson, “Multimodel inference: Understanding AIC and BIC in model selection,” *Sociological Methods and Research*, vol. 33, no. 2. pp. 261–304, Nov. 2004, doi: 10.1177/0049124104268644.

[129] M. Rosenblatt, “Remarks on a Multivariate Transformation,” *Ann. Math. Stat.*, vol. 23, no. 3, pp. 470–472, 1952, doi: 10.1214/aoms/1177729394.

[130] K. Aas, C. Czado, A. Frigessi, and H. Bakken, “Pair-copula constructions of multiple dependence,” *Insur. Math. Econ.*, vol. 44, no. 2, pp. 182–198, 2009, doi: 10.1016/j.insmatheco.2007.02.001.

[131] C. E. Tuncali, S. Yaghoubi, T. P. Pavlic, and G. Fainekos, “Functional gradient descent optimization for automatic test case generation for vehicle controllers,” in *IEEE International Conference on Automation Science and Engineering*, 2017, vol. 2017-Augus, pp. 1059–1064, doi: 10.1109/COASE.2017.8256245.

[132] C. E. Tuncali, G. Fainekos, D. Prokhorov, H. Ito, and J. Kapinski, “Requirements-driven Test Generation for Autonomous Vehicles with Machine Learning Components,” *IEEE Trans. Intell. Veh.*, pp. 1–1, 2019, doi: 10.1109/tiv.2019.2955903.

[133] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017, doi: 10.1145/3065386.

[134] A. Hannun *et al.*, “Deep Speech: Scaling up end-to-end speech recognition,” *arxiv.org*,

2014, Accessed: Mar. 09, 2020. [Online]. Available: https://arxiv.org/abs/1412.5567.

[135] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, 2014, vol. 4, no. January, pp. 3104–3112, Accessed: Mar. 09, 2020. [Online]. Available: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-.

[136] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial Patch," no. Nips, 2017, [Online]. Available: http://arxiv.org/abs/1712.09665.

[137] N. Papernot, P. Mcdaniel, and I. Goodfellow, "Practical Black-Box Attacks against Machine Learning," 2017.

[138] S. Thys, W. Van Ranst, and T. Goedeme, "Fooling automated surveillance cameras: Adversarial patches to attack person detection," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, Apr. 2019, vol. 2019-June, pp. 49–55, doi: 10.1109/CVPRW.2019.00012.

[139] E. R. Balda, A. Behboodi, and R. Mathar, "Adversarial examples in deep neural networks: An overview," in *Studies in Computational Intelligence*, vol. 865, Springer Verlag, 2020, pp. 31–65.

[140] C. Xiao *et al.*, "Characterizing Attacks on Deep Reinforcement Learning." Accessed: Mar. 09, 2020. [Online]. Available: https://arxiv.org/abs/1907.09470.

[141] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, "Adversarial Attacks on Neural Network Policies," *5th Int. Conf. Learn. Represent. ICLR 2017 - Work. Track Proc.*, 2019, Accessed: Mar. 09, 2020. [Online]. Available: http://rll.berkeley.edu/adversarial.

[142] Y. Lin, Z. Hong, Y. Liao, M. Shih, M. Liu, and M. Sun, "Tactics of Adversarial Attack on Deep Reinforcement Learning Agents," 2017.

[143] Tencent Keen Security Lab, "Experimental Security Research of Tesla Autopilot," p. 38, 2019, [Online]. Available: https://keenlab.tencent.com/en/whitepapers/Experimental_Security_Research_of_Tesla_Autopilot.pdf.

[144] L. S. Shapley, "Stochastic Games," in *Stochastic Games and Applications*, 2003, pp. 1–7.

[145] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a Formal Model of Safe and Scalable Self-driving Cars," pp. 1–37, 2017, doi: 1708.06374v2.

[146] A. Shashua, S. Shalev-Shwartz, and S. Shammah, "Implementing the RSS Model on NHTSA Pre-Crash Scenarios," 2018.

[147] "中华人民共和国道路交通安全法." http://www.gov.cn/banshi/2005-08/23/content_25575.htm (accessed May 18, 2020).

[148] "TRANSPORTATION CODE CHAPTER 545. OPERATION AND MOVEMENT OF VEHICLES." https://statutes.capitol.texas.gov/docs/TN/htm/TN.545.htm (accessed May 18, 2020).

[149] "Article 25 Vehicle Law | Driving Right Overtaking Passing." http://ypdcrime.com/vt/article25.htm#t1128 (accessed May 18, 2020).

[150] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, pp. 1–21, 2016.

[151] C. Kanbak, S. M. Moosavi-Dezfooli, and P. Frossard, “Geometric Robustness of Deep Networks: Analysis and Improvement,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 4441–4449, 2018, doi: 10.1109/CVPR.2018.00467.

[152] M. Liu, S. Liu, H. Su, K. Cao, and J. Zhu, “Analyzing the Noise Robustness of Deep Neural Networks,” *2018 IEEE Conf. Vis. Anal. Sci. Technol. VAST 2018 - Proc.*, no. October, pp. 60–71, 2018, doi: 10.1109/VAST.2018.8802509.

[153] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine, “EPOpt: Learning Robust Neural Network Policies Using Model Ensembles,” Oct. 2016, Accessed: Nov. 14, 2020. [Online]. Available: http://arxiv.org/abs/1610.01283.

[154] F. Muratore, F. Treede, M. Gienger, and J. Peters, “Domain Randomization for Simulation-Based Policy Optimization with Transferability Assessment,” *2nd Conf. Robot Learn. (CoRL 2018)*, no. CoRL, pp. 1–14, Oct. 2018, Accessed: Nov. 14, 2020. [Online]. Available: http://proceedings.mlr.press/v87/muratore18a.html.

[155] V. Tjeng, K. Xiao, and R. Tedrake, “Evaluating robustness of neural networks with mixed integer programming,” *arXiv*. arXiv, Nov. 20, 2017, Accessed: Nov. 14, 2020. [Online]. Available: http://arxiv.org/abs/1711.07356.

[156] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, “Reluplex: An efficient smt solver for verifying deep neural networks,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2017, vol. 10426 LNCS, pp. 97–117, doi: 10.1007/978-3-319-63387-9_5.

[157] A. Lomuscio and L. Maganti, “An approach to reachability analysis for feed-forward ReLU neural networks,” *arXiv*. arXiv, Jun. 22, 2017, Accessed: Nov. 14, 2020. [Online]. Available: http://arxiv.org/abs/1706.07351.

[158] F. Chang, P. Xu, H. Zhou, J. Lee, and H. Huang, “Identifying motorcycle high-risk traffic scenarios through interactive analysis of driver behavior and traffic characteristics,” *Transp. Res. Part F Traffic Psychol. Behav.*, vol. 62, pp. 844–854, Apr. 2019, doi: 10.1016/j.trf.2019.03.010.

[159] M. Botvinick, S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell, and D. Hassabis, “Reinforcement Learning, Fast and Slow,” *Trends in Cognitive Sciences*, vol. 23, no. 5. Elsevier Ltd, pp. 408–422, May 01, 2019, doi: 10.1016/j.tics.2019.02.006.

[160] A. Nagabandi *et al.*, “Learning to adapt in dynamic, real-world environments through meta-reinforcement learning,” *arXiv*, pp. 1–17, 2018, Accessed: Nov. 20, 2020. [Online]. Available: https://sites.google.com/berkeley.edu/metaadaptivecontrol.

[161] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine, “One-shot visual imitation learning via meta-learning,” *arXiv*. 2017, Accessed: Nov. 20, 2020. [Online]. Available: https://arxiv.org/abs/1709.04905.

[162] T. Yu *et al.*, “One-Shot Imitation from Observing Humans via Domain-Adaptive Meta-Learning.” Accessed: Nov. 20, 2020. [Online]. Available: https://sites.google.com/view/daml.

[163] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, “RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning,” 2016, Accessed: Aug. 19, 2020. [Online]. Available: http://arxiv.org/abs/1611.02779.

[164] J. X. Wang *et al.*, “Learning to reinforcement learn,” 2016, Accessed: Nov. 21, 2020. [Online]. Available: http://arxiv.org/abs/1611.05763.

[165] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, “Memory-based control with recurrent neural networks,” 2015, Accessed: Nov. 21, 2020. [Online]. Available: http://arxiv.org/abs/1512.04455.

[166] C. Finn, P. Abbeel, and S. Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” *34th Int. Conf. Mach. Learn. ICML 2017*, vol. 3, pp. 1856–1868, 2017.

[167] J. Rothfuss, D. Lee, I. Clavera, T. Asfour, and P. Abbeel, “PROMP: Proximal Meta-Policy Search,” *arXiv*. Oct. 15, 2018, Accessed: Nov. 21, 2020. [Online]. Available: http://arxiv.org/abs/1810.06784.

[168] A. Gupta, R. Mendonca, Y. X. Liu, P. Abbeel, and S. Levine, “Meta-reinforcement learning of structured exploration strategies,” *arXiv*. 2018, Accessed: Nov. 21, 2020. [Online]. Available: http://papers.nips.cc/paper/7776-meta-reinforcement-learning-of-structured-exploration-strategies.

[169] K. Rakelly, A. Zhou, D. Quiilen, C. Finn, and S. Levine, “Efficient off-policy meta-reinforcement learning via probabilistic context variables,” in *36th International Conference on Machine Learning, ICML 2019*, 2019, vol. 2019-June, pp. 9291–9301, Accessed: Aug. 05, 2020. [Online]. Available: https://github.com/katerakelly/oyster.

[170] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor,” 2018, [Online]. Available: http://arxiv.org/abs/1801.01290.

[171] E. Leurent, “An environment for autonomous driving decision-making,” *Github*, 2018. https://github.com/eleurent/highway-env.

[172] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A Next-generation Hyperparameter Optimization Framework,” *dl.acm.org*, pp. 2623–2631, Jul. 2019, doi: 10.1145/3292500.3330701.

[173] J. Schulman *et al.*, “Trust Region Policy Optimization,” 2015, doi: 10.1080/03004279.2014.996242.

[174] C. Finn, A. Rajeswaran, S. Kakade, and S. Levine, “Online meta-learning,” *36th Int. Conf. Mach. Learn. ICML 2019*, vol. 2019-June, pp. 3398–3410, 2019.