Securing Autonomous Driving: Addressing Adversarial Attacks and Defenses

Jinghan Yang

Washington University – McKelvey School of Engineering

Follow this and additional works at: https://openscholarship.wustl.edu/eng_etds

Recommended Citation
https://openscholarship.wustl.edu/eng_etds/951

This Dissertation is brought to you for free and open access by the McKelvey School of Engineering at Washington University Open Scholarship. It has been accepted for inclusion in McKelvey School of Engineering Theses & Dissertations by an authorized administrator of Washington University Open Scholarship. For more information, please contact digital@wumail.wustl.edu.
WASHINGTON UNIVERSITY IN ST. LOUIS
McKelvey School of Engineering
Department of Computer Science & Engineering

Dissertation Examination Committee:
Christopher Gill, Chair
Ayan Chakrabarti
Yevgeniy Vorobeychik
Ning Zhang
Silvia Zhang

Securing Autonomous Driving: Addressing Adversarial Attacks and Defenses
by
Jinghan Yang

A dissertation presented to
the McKelvey School of Engineering
of Washington University in
partial fulfillment of the
requirements for the degree
of Doctor of Philosophy

August 2023
St. Louis, Missouri
© 2023, Jinghan Yang
# Table of Contents

List of Figures ................................................................. v  
List of Tables ................................................................. vii  
Acknowledgments ............................................................. ix  
Abstract ................................................................. x  

I Introduction and Background ............................................. 1  

Chapter 1: Introduction ....................................................... 2  
1.1 Understanding and Addressing Security Challenges in Autonomous Driving Models .................................................. 7  
1.2 Contributions .......................................................... 9  

Chapter 2: Literature Review: Patch Attacks and Defense Strategies for Image Recognition Models ........................................... 11  
2.1 Patch Attack Definition ................................................ 12  
2.2 Patch Attacks for Image Classification and Object Detection .................................................. 15  
2.3 Adversarial Patch Detection and Defense ................................ 18  

Chapter 3: Literature Review: Malicious Attacks and Defense Mechanisms for Autonomous Driving System and Connected Autonomous Driving Fleet .................................................. 22  
3.1 Adversarial Manipulation of Sensor Inputs in Autonomous Driving: Attacking the Perception Module .................................................. 23  
3.2 Adversarial Attacks on Localization in Autonomous Driving .................................................. 25  
3.3 Defense Mechanisms for Enhancing Robustness of Autonomous Driving System against Malicious Attacks .................................................. 27  

Chapter 4: Dataset and Simulation Environment for Image Recognition and Autonomous Driving Research ........................................... 28
# The Vulnerabilities of Autonomous Driving System and Connected Autonomous Driving Fleet

## Chapter 5: A Fast and Differentiable Adversarial Testing Framework for Simulated Autonomous Driving

5.1 Introduction ............................................. 32
5.2 Proposed Method .......................................... 34
  5.2.1 Parameterized Scene Modifications .................... 35
  5.2.2 Approximate Frames via Compositing .................. 36
  5.2.3 Computing Adversarial Perturbations .................. 37
5.3 Experiments ............................................ 39
  5.3.1 Attack Optimization .................................... 41
  5.3.2 Results ................................................ 41
  5.3.3 Visualization ......................................... 46

## Chapter 6: Location Spoofing Attacks on Autonomous Fleets

6.1 Introduction ............................................. 48
6.2 Additional Related Work ................................... 50
6.3 Model .................................................. 52
  6.3.1 Hardness .............................................. 54
  6.3.2 Approach: Design of GPS Spoofing Attacks on Autonomous Fleets . 55
  6.3.3 Experiments ........................................... 61
  6.3.4 Service Failure Attack ................................ 66

# Defense Mechanism: Developing Robust Image Recognition Model and Autonomous Driving System

## Chapter 7: Patch Defense via Contrastive Adversarial Semantic Meaning

7.1 Introduction ............................................. 71
7.2 Additional Related Work ................................... 72
7.3 Preliminaries ........................................... 73
7.4 Method ................................................ 75
  7.4.1 Patch Ablation ....................................... 76
  7.4.2 Adversarial Semantic Meaning ........................ 76
  7.4.3 Defense .............................................. 77
  7.4.4 Adversarial Training .................................. 80
  7.4.5 Evasive Attacks ...................................... 83
7.5 Experiment ............................................ 84
  7.5.1 Results ............................................... 85

## Chapter 8: Certified Robust Control under Adversarial Perturbations

8.1 Introduction ............................................. 89
List of Figures

Figure 5.1: Overview. We collect and calibrate frames from the unmodified environment (shown in the green box), and given a choice of attack pattern parameters, composite the pattern to create approximate renderings of frames corresponding to placing the pattern in the environment. Our composition function is differentiable with respect to the attack pattern parameters, and we are thus able to use end-to-end gradient-based optimization when attacking a differentiable control network, to cause the network to output incorrect controls that cause the vehicle to deviate from its intended trajectory (from the green to the blue trajectory, as shown in the right column), and crash. .................................................. 33

Figure 5.2: Trajectory deviations induced by GradOpt and BO for 6 example scenarios. ................................................................. 43

Figure 5.3: Frames from driving simulations, without cars or pedestrians in different weather conditions, after introducing attack patterns from GradOpt (top) and BO (bottom). .................................................. 47

Figure 6.1: Routes from starting location (circle) to destination (star) of two fleet vehicles with no spoofing (green and blue) and with 4 spoofing devices (red and orange) in OpenStreetView. The traffic graph (radius 1000) is marked in black; spoofing locations and effects are marked as purple arrows. . . 50
Figure 7.1: (AdvSemShield) Input images are processed by the detector $h$ which identifies and ablates adversarial patches. These ablations are passed to a 'Patch Branch' which produces a semantic embedding ($z_p$). Concurrently, the background image is processed by the 'Background Branch'. These embedding are combined for prediction. At training time, the Patch Branch and Background Branch are first trained separately with supervised contrastive loss $L_c$ and the cross-entropy loss $L_B$ respectively (green and pink arrows). The pipeline then undergoes end-to-end training, which employs a weighted combination of three losses, including cross entropy loss of the models final predictions $L_D$ (blue arrows).

Figure 7.2: Semantic embedding of ablated patches. Model $g$ gives the adversarial class of each patch (e.g., the first dog is predicted as a cat). Contrastive learning is used to embed ablated patches into the latent space; positive and negative examples are defined via their adversarial class (Definition 7.4.1).

Figure 7.3: Distribution of adversarial classes for images with true class cat in CIFAR 10.
List of Tables

Table 5.1: Ablation analysis of variations of GradOpt. ................................................. 42
Table 5.2: Average deviation and infraction penalties over all scenarios for GradOpt and BO, when optimizing parameters of different numbers of rectangles (1-b optimizes only the shape of one black rectangle) in “clear noon” weather. The % ≥ column reports the percentage of instances where GradOpt has ≥ score than BO. ................................................................. 43
Table 5.4: Infraction penalties without cars or pedestrians, i.e., infraction penalties computed with only static objects, in standard “clear noon” simulations for each type of scenario. ................................................................. 45
Table 5.5: Infraction penalties over all scenarios with weather conditions different from that used for optimizing attacks (“clear noon”). ................................................. 45
Table 5.6: Deviations in simulations over all scenarios with weather conditions different from that used for optimizing adversarial patterns (“clear noon”). ................................................. 45
Table 5.7: Infraction penalties in simulations without cars or pedestrians over all scenarios with weather conditions different from that used for optimizing attacks (“clear noon”). ................................................. 46
Table 6.1: The average travel time in unmodified driving environment and delay ratio induced by spoofers in the static-static case in a dist-1000 meters traffic network. The travel time unit is second. ................................................. 62
Table 6.2: The average travel time in unmodified driving environment and delay ratio induced by spoofers in the static-dynamic case in a dist − 500 traffic network. ................................................. 64
Table 6.3: The average travel time in unmodified driving environment and delay ratio induced by spoofers in the static-dynamic case in a dist − 1000 traffic network. ................................................. 65
Table 6.4: The delay ratio in the \textit{dynamic-dynamic} case in a \textit{dist} – 500 traffic network with induced by spoof devices with spoofing radius 1. .......................... 65

Table 6.5: The delay ratio in the \textit{dynamic-dynamic} case in a \textit{dist} – 1000 traffic network induced by spoof devices with spoofing radius 1. .......................... 65

Table 6.6: The service failure ratio induced by spoofing devices with spoofing radius 1 in the \textit{static-static} case in a \textit{dist-500} traffic network. .......................... 68

Table 6.7: The service failure ratio induced by spoofing devices with spoofing radius 1 in the \textit{static-static} case in a \textit{dist-1000} traffic network. .......................... 68

Table 6.8: The service failure ratio induced by spoofers in the \textit{static-dynamic} case in a \textit{dist-500} traffic network. .......................... 69

Table 6.9: The service failure ratio induced by spoofers in the \textit{static-dynamic} case in a \textit{dist-1000} traffic network. .......................... 69

Table 7.1: Model robustness across different datasets, patch ratios, and evation terms. .......................... 85

Table 7.2: Clean accuracy .......................... 85

Table 7.3: Accuracy of a 2-layer fully connected network which uses the latent space of the patch-branch to predict the adversarial label (Definition 7.4.1.) .......................... 86

Table 7.4: True Positive Rate of the detector used in each method. For PatchZero and AdvSemShield the False Positive Rate of the detector is less than 0.01 in all cases. .......................... 87

Table 8.1: Accuracy and performance without malicious attacks .......................... 101

Table 8.2: Vulnerability of the non-robust perception model \( f \) .......................... 101

Table 8.3: Robustness: Instability (up) Efficiency (bottom) under attacks. .......................... 102

Table 8.4: Robustness: instability and efficiency .......................... 102

Table 8.5: The MSE and performance of road friction regression in benign (up) and adversarial (bottom) environment. .......................... 103
Acknowledgments

I am thankful to my family for their unwavering support, love, and belief in my abilities during my PhD journey. Their presence has been the driving force behind my accomplishments.

This work was supported by National Science Foundation.

Jinghan Yang

Washington University in St. Louis
August 2023
ABSTRACT OF THE DISSERTATION

Securing Autonomous Driving: Addressing Adversarial Attacks and Defenses

by

Jinghan Yang

Doctor of Philosophy in Computer Science
Washington University in St. Louis, 2023

Professor Christopher Gill, Chair

 Autonomous driving systems have gained significant attention in recent years, revolutionizing the transportation industry. However, the increasing complexity and connectivity of these systems introduce new challenges and vulnerabilities. This thesis addresses these challenges by investigating adversarial attacks and defenses in the field of autonomous driving.

The thesis commences with a comprehensive literature review, providing insights into the vulnerabilities unique to image recognition and autonomous driving. It examines the current research lines for defense strategies and explores the ongoing efforts to mitigate these vulnerabilities.

In the subsequent sections, the thesis presents four core contributions in adversarial attacks and defenses. Section II focuses on the vulnerabilities of autonomous driving systems and connected autonomous driving fleets. It proposes a fast and differentiable adversarial testing framework for simulated autonomous driving, demonstrating its scalability and effectiveness in identifying vulnerabilities. Additionally, the systemic impact of GPS spoofing attacks on large-scale autonomous vehicle deployment in the context of ride-hailing services is investigated. The research explores innovative approaches to mitigate the risks associated with spoofing devices.
Moving forward, Section III delves into defense mechanisms against adversarial attacks. It introduces a novel defense approach to counter adversarial patch attacks in image classification, leveraging contrastive adversarial semantic meaning. Furthermore, the thesis addresses the challenge of maintaining robustness in machine learning-based control systems under adversarial perturbations. It proposes a certified robust control approach that combines robustness certification with control, resulting in a certified robust autonomous driving system.

Overall, this thesis contributes to the understanding of adversarial vulnerabilities in autonomous driving systems and provides valuable insights into the development of robust defenses. The findings pave the way for enhancing the security and reliability of autonomous driving technologies, ensuring their safe deployment in real-world scenarios.
Part I

Introduction and Background
Chapter 1

Introduction

Autonomous driving, powered by artificial intelligence technologies, has garnered significant attention in academia and industry. Over the past two decades, significant advancements have been made in autonomous driving systems (ADSs), resulting in various levels of automation, ranging from level 0 (no automation) to level 4 (high self-driving automation). Companies like Tesla are focused on developing level 3 ADSs, which offer limited self-driving capabilities under specific conditions, such as on highways. On the other hand, Google Waymo is dedicated to researching and industrializing level 4 ADSs, aiming for operation without human intervention in most situations. Autonomous vehicles are expected to greatly enhance the driving experience, but critical issues, particularly related to safety, need to be addressed before widespread industrialization can occur. Deep learning, a popular AI technique, plays a crucial role in autonomous vehicles for perception tasks and real-time decision making. The workflow and architecture of a deep learning-based ADS involve multiple stages. In the perception layer, raw data from various sensors and high-definition (HD) map information are fed into deep learning models to extract environmental information. The decision layer utilizes designated deep learning models to make real-time decisions based on the extracted information. For instance, Baidu Apollo, Tesla’s Autopilot, and Waymo employ deep learning models for perception and object detection tasks.

In this thesis, we focus on deep learning-based autonomous driving system. This type of system is normally composed of three functional layers, including a sensing layer, a perception layer and a decision layer, as well as an additional cloud and service layer. An autonomous driving car often attached with several heterogeneous sensors such as GPS, camera, LiDAR, radar, and ultrasonic sensors. These sensors are used to collect physical environment information in real-time, for examples, driving frames, point clouds, location etc.

An autonomous driving car in the high-level has few modules. In a deep learning based ADS, the perception module contains deep learning models to analyze the data collected by the
sensing layer and then extract useful environmental information from the raw data for further process. For example, detecting the traffic light, pedestrians. Next, the decision layer would act as a decision-making unit to output instructions concerning the change of speed and steering angle based on the extracted information from the perception layer.

The most preferred sensors adopted and deployed by leading autonomous driving vehicle companies like Waymo are GPS/Inertial Measurement Units (IMU), cameras, Light Detection and Ranging (Lidar), Radio Detection and Ranging (Radar), and ultrasonic sensors. Cameras are often used to capture visual information around an autonomous vehicle, providing abundant information to the perception module to analyze surroundings. Furthermore, Radar and Lidar plays more and more important roles in the modern self-driving cars. They emit lights to the surroundings. The lights will be reflected by other objects. Therefore, point clouds will be established by measuring distances between objects and the vehicle based on the reflection of light. More specifically, Lidar are often being used for detecting objects in the long distance, and Radar are more accurate at observing the environment nearby. Note that, as the development of Radar, modern Radar are more and more being used for more tasks. Note that, the efficacy of Lidar can decrease by the weather, and Radar are more robust for different weathers. Lidar and Radar are used for object detection and localization. GPS is used for localization and IMU provides orientation, velocity and acceleration data.

Roughly speaking, the autonomous driving system can be divided into three main modules: the perception module, the localization module, and the decision module.

**Perception Module** The perception module in autonomous driving systems (ADSs) plays a crucial role in processing raw sensory data and extracting traffic information. This includes tasks such as traffic light recognition, object detection and panoptic segmentation. Road object detection is particularly challenging due to the complex nature of identifying various objects, including lanes, traffic signs, vehicles, and pedestrians, in real-time and dynamic environments. In the field of object detection, Faster RCNN [47] is widely recognized for its effectiveness in detecting objects in images. Another notable algorithm is You Only Look Once (YOLO) [119], which approaches the detection task as a regression problem. Additionally, there has been significant research and industry focus on Lidar-based object detection models. VoxelNet [174], for instance, is an end-to-end model that directly predicts objects based on Lidar point cloud data. PointRCNN [130], on the other hand, adapts the RCNN architecture
for object detection using 3D point cloud input, achieving superior performance. In summary, the perception module in ADSs employs various deep learning modules to process raw sensory data and extract valuable traffic information. Object detection algorithms like Faster RCNN and YOLO are effective in detecting objects, while LiDAR-based models such as VoxelNet and PointRCNN provide superior performance in handling 3D point cloud data. For semantic segmentation, FCN and PSPNet are notable models that excel in classifying image regions into specific classes.

### Localization Module

The Localization Module is a critical component of autonomous driving systems (ADSs) that plays a key role in route planning. Its primary function is to accurately determine the vehicle’s location on the map and provide an understanding of the real-time environment. This module combines data from various sources, including GPS, IMU, Lidar point clouds, and HD maps, to achieve precise localization.

The process of localization involves fusing the data from these sources to perform tasks such as odometry estimation and map reconstruction. Odometry estimation focuses on analyzing the sensor data to determine the vehicle’s movement accurately. By considering factors such as speed, direction, and rotation, the module can estimate the vehicle’s position relative to its starting point. Map reconstruction tasks involve creating a detailed map of the surrounding environment based on the sensor inputs. This helps in enhancing the understanding of the vehicle’s surroundings and aids in accurate localization.

Researchers have explored different techniques to improve the localization capabilities of ADSs. In one study [146], a combination of Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) was used to estimate the vehicle’s movement and poses. By analyzing continuous images captured by a camera, these deep learning architectures were able to extract relevant information for precise localization. Another approach [5] involved utilizing a deep autoencoder, which encoded observed images into a compressed representation. This compressed format was then utilized for map reconstruction and localization purposes. This method effectively reduced the dimensionality of the image data, resulting in efficient map construction and improved localization accuracy.

Additionally, the use of High-Definition Maps (HD Maps) has emerged as a valuable tool in the localization module. HD Maps provide detailed and accurate information about the road environment, including lane markings, road geometries, traffic signs, traffic lights, and speed
limits. These maps, created using advanced surveying techniques such as Lidar scanning and camera imaging, serve as a point of reference for the vehicle’s localization and path planning. By comparing real-time sensor data from cameras, Lidar, and other sensors with the information stored in the HD Map, the autonomous vehicle can precisely determine its position on the road and gain a comprehensive understanding of the driving environment.

In conclusion, the Localization Module in ADSs combines data from GPS, IMU, Lidar, and HD maps to achieve accurate localization. Odometry estimation and map reconstruction are vital tasks within this module, and the integration of CNNs, RNNs, and deep autoencoders enhances the localization performance. The inclusion of HD Maps provides a valuable reference for precise localization and aids in comprehensive path planning in autonomous driving systems.

**Decision Module**  The decision module is a critical component in an Autonomous Driving System (ADS) that is responsible for making high-level decisions and controlling the vehicle’s behavior based on the perceived environment. It plays a crucial role in ensuring the safe and efficient operation of autonomous vehicles.

The decision module encompasses various tasks, including path planning, object trajectory prediction, vehicle control, and end-to-end driving. These tasks involve complex decision-making processes that rely on the integration of sensor data, perception algorithms, and deep learning models.

Path Planning and Object Trajectory Prediction: Path planning is a fundamental task in autonomous vehicles, involving determining the optimal route from a start location to a desired destination. Meanwhile, object trajectory prediction requires vehicles to anticipate the movement of perceived obstacles based on sensor data and perception. In recent studies, Inverse Reinforcement Learning has shown promising results in path planning by learning reward functions from human drivers, enabling the vehicle to generate routes that align with human driving behavior [51]. For trajectory prediction, variations of Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) models have been proposed to achieve accurate and efficient predictions. Luo et al. explored the use of 3D spatial-temporal data and a single Convolutional Neural Network (CNN) for car trajectory forecasting [94].
Vehicle control is an essential aspect of the decision module, where the ADS uses algorithms and models to control the vehicle's movements. Traditional rule-based algorithms may not adequately address the complexities of real-world driving scenarios. As a result, deep reinforcement learning techniques have emerged as promising approaches for training autonomous agents to make informed decisions in various driving situations.

Broadly speaking, there are two main types of decision modules: end-to-end models and models that integrate high-level decision-making with low-level controllers.

End-to-End Driving: End-to-End driving models integrate perception and decision-making processes into a single deep learning model. In this setup, the model predicts the current steering angle and driving speed based on ambient sensory information. For instance, the DAVE-2 system introduced an E2E driving model based on a CNN architecture, using front-facing camera images as input to predict the steering angle [6].

Combination with a Low-Level Controller: To enhance the overall driving performance, combining the high-level decision-making models with a low-level controller is crucial. The low-level controller executes the commands generated by the decision-making module, translating them into specific control actions for the vehicle. This integration ensures the smooth execution of the planned trajectory and enhances the vehicle’s overall control capabilities.

This thesis explores two types of autonomous driving systems (ADSs) that differ in their approach to integrating perception and decision-making:

1) Vision-based end-to-end ADS: In this approach, perception and decision-making are combined as a single integrated system. The ADS takes raw sensor data as input and directly generates the appropriate driving actions without explicitly separating the perception and decision processes. This end-to-end fashion aims to learn complex mappings from input sensor data to desired driving actions using deep learning techniques. By leveraging large amounts of data and neural network architectures, end-to-end ADSs attempt to learn a comprehensive representation of the entire perception-to-action pipeline.

2) Vision-based perception module with high-level decision and low-level controller: In this approach, the perception module focuses on extracting relevant information from visual sensor data, such as cameras, to understand the driving environment. This perception module, typically based on deep learning models, processes the visual input to detect objects,
recognize road features, and extract relevant contextual information. The output from the perception module is then combined with a high-level decision-making deep learning model. This high-level decision model interprets the perception data and generates appropriate driving decisions, such as route planning and object avoidance. Finally, these decisions are implemented by a low-level controller that directly interacts with the vehicle’s actuators, such as steering, acceleration, and braking systems.

1.1 Understanding and Addressing Security Challenges in Autonomous Driving Models

Deep learning has been shown to be vulnerable to malicious attacks. The rapid development and adoption of autonomous systems have revolutionized various domains, including image recognition, autonomous driving, and connected autonomous driving fleets. Nonetheless, the increasing reliance on these systems has also raised concerns about their security and reliability in the face of adversarial attacks. Adversarial attacks aim to exploit vulnerabilities in autonomous systems and can have serious consequences, such as misclassification, navigation errors, and potential risks to passenger safety. Therefore, it is crucial to develop effective defense mechanisms to mitigate these attacks and ensure the robustness of autonomous systems.

This thesis aims to explore the vulnerabilities of different autonomous driving systems (ADSs) and address the challenges posed by adversarial attacks. It focuses on two main approaches: the end-to-end approach and the combination of vision-based perception with high-level decision-making and a low-level controller.

The first domain of investigation is image recognition, where the thesis examines the vulnerabilities of ADSs in recognizing and processing visual information. Adversarial attacks in this domain aim to deceive the perception module by manipulating images or adding subtle perturbations that can mislead the image recognition models. The thesis explores these vulnerabilities and proposes robust defense strategies to enhance the security and reliability of image recognition systems in ADSs. These defense strategies may include techniques such as adversarial training, defensive distillation, or the integration of anomaly detection mechanisms to detect and mitigate adversarial attacks.
The second domain of investigation is autonomous driving vehicles. Here, the thesis analyzes the vulnerabilities in the decision-making process of ADSs. Adversarial attacks in this domain can manipulate the decision module, causing the vehicle to make incorrect or unsafe driving decisions. By studying these vulnerabilities, the thesis aims to develop effective defense mechanisms to ensure the integrity and safety of decision-making in autonomous driving systems. These defense mechanisms may involve techniques such as anomaly detection, reinforcement learning with safety constraints, or the integration of multiple decision-making models for robustness.

The third domain of investigation focuses on connected autonomous driving fleets. This involves studying the vulnerabilities that arise when multiple autonomous vehicles are connected in a fleet, sharing information and coordinating their actions. Adversarial attacks in this domain can disrupt the communication and coordination among vehicles, leading to potential safety risks. The thesis aims to identify these vulnerabilities and propose robust defense strategies to protect connected autonomous driving fleets from adversarial attacks. These defense strategies may include secure communication protocols, anomaly detection in data sharing, or the integration of redundancy and fault-tolerant mechanisms to mitigate the impact of attacks.

By considering these three distinct domains and investigating the vulnerabilities and challenges in ADSs, the thesis aims to contribute to the development of robust defense strategies against adversarial attacks. The goal is to enhance the security, reliability, and safety of autonomous driving systems, ultimately advancing the adoption and deployment of autonomous vehicles in real-world scenarios.

Deep learning models have demonstrated impressive accuracy across various domains, but they are not robust against adversarial attacks. These attacks exploit the susceptibility of deep learning models by introducing subtle modifications to input images, which can cause misclassification and compromise the reliability of the models. Recognizing the significance of this challenge, my doctoral research is primarily focused on addressing the vulnerabilities introduced by adversarial attacks in deep learning models and developing robust solutions to mitigate their impact. After extensively reviewing the existing literature on the vulnerabilities and robustness of image recognition and autonomous driving, sections II and III of my thesis will focus on presenting the contributions of my doctoral research. In summary, my doctoral research has made the following contributions:
1.2 Contributions

In this thesis, we have addressed various challenges and vulnerabilities in autonomous driving systems and connected autonomous driving fleets. We began by conducting a literature review that provided insights into the vulnerabilities specifically related to image recognition and autonomous driving. This literature overview served as the foundation for our subsequent research. In Section II and Section III of this thesis, we present the core contributions of my PhD research, which focus on addressing adversarial attacks and defenses in the field of autonomous driving.

In Section II, we focused on exploring the vulnerabilities of autonomous driving systems and connected autonomous driving fleets. We explored the impact of adversarial attacks on perception architectures and proposed a fast and differentiable adversarial testing framework for simulated autonomous driving in chapter 5. Our framework demonstrated its scalability and effectiveness in identifying vulnerabilities in simulated experiments compared to state-of-the-art methods. Next, Chapter 6 investigates the systemic impact of GPS spoofing on large-scale autonomous vehicle deployment in the context of ride-hailing services. While GPS spoofing attacks targeting individual vehicles have been studied, our goal is to understand the broader implications of a limited number of strategically placed spoofing devices on the quality of the entire ride-hailing service. We explore two variants of the problem: a static variant with fixed spoofing device locations and configurations, and a dynamic variant where both the devices and their configurations can change over time. Additionally, we consider two attack objectives: maximizing overall travel delay and minimizing the number of successfully completed ride requests. To solve this NP-hard problem, we present an integer linear programming approach for the static variant and a novel deep reinforcement learning approach for the dynamic variant. Through experiments conducted on a real traffic network, we demonstrate the significant impact of even a few spoofing devices on the efficacy of an autonomous ride-hailing fleet, highlighting the potential risks associated with large-scale deployment.

In Section III, we introduced two notable publications that focused on defense mechanisms against adversarial attacks. In Chapter 7, we presented a novel defense approach specifically designed to counter adversarial patch attacks in the context of image classification. Our approach leveraged contrastive adversarial semantic meaning, which significantly enhanced the robustness of models and achieved superior performance compared to existing baseline
methods. By effectively analyzing attack strategies and identifying the semantic meaning of adversarial patches, we were able to accurately classify images even in the presence of subtle adversarial perturbations. Moving on to Chapter 8, we addressed the challenge of maintaining robustness in machine learning-based control systems when subjected to adversarial perturbations. In this publication, we proposed a certified robust control approach that integrated robustness certification with control, ultimately leading to the development of a certified robust autonomous driving system. By combining the certification of predictions with control mechanisms, we ensured the system’s ability to handle adversarial perturbations while maintaining reliable and safe control over the vehicle. Extensive evaluations and assessments were conducted to gauge the end-to-end certified robustness of the integrated vision and control components, demonstrating the significance and effectiveness of the obtained certificates in reinforcing the system’s overall robustness and performance. These two publications contribute to the field of adversarial defenses in autonomous driving by introducing innovative and effective approaches to counter adversarial attacks. Through extensive experimentation and evaluation, we demonstrate the practicality and efficacy of our proposed defense mechanisms, showcasing their potential for enhancing the security and reliability of autonomous driving systems.

Through our research, we have made significant contributions to the field of adversarial attacks and defenses in autonomous driving systems. We have identified vulnerabilities, proposed innovative approaches, and conducted extensive experiments using various datasets and simulation environments. These contributions pave the way for more secure and robust autonomous driving systems in the future.

In conclusion, this thesis has shed light on the vulnerabilities, challenges, and potential solutions related to adversarial attacks in autonomous driving systems. Our findings and proposed defense mechanisms provide valuable insights for researchers and practitioners in the field. With the continuous advancements in autonomous driving technology, it is crucial to address these security concerns and ensure the safety and reliability of autonomous vehicles in real-world scenarios.
Chapter 2

Literature Review: Patch Attacks and Defense Strategies for Image Recognition Models

This chapter serves as a comprehensive literature review on patch attacks and defense strategies for image recognition models. Patch attacks are a specific type of adversarial attack that involves adding visually inconspicuous patches to images, aiming to deceive the image recognition system. These attacks exploit vulnerabilities in the decision-making process of the model, leading to incorrect interpretation of images and potential security risks.

The chapter begins by introducing the concept of patch attacks and emphasizing their significance within the context of adversarial attacks on image recognition models. The motivations behind patch attacks are explored, along with their potential impact on various applications that heavily rely on image recognition technology.

The subsequent section focuses on the related work pertaining to patch attacks. Seminal studies, including the notable work by Brown et al. [9], which initially demonstrated the concept of adversarial patch attacks, are reviewed. This section delves into various optimization methods and techniques employed to generate effective patches that remain visually imperceptible to humans. Additionally, the success rates and limitations of these attack methods are examined, shedding light on their practical implications.

Lastly, the chapter delves into the literature concerning defense strategies against patch attacks. It surveys a range of approaches and techniques proposed by researchers to bolster the robustness of image recognition models against patch attacks. These defense strategies aim to detect and mitigate the impact of adversarial patches, thereby improving the model’s ability to accurately classify images even in the presence of such attacks.
In summary, this chapter provides a comprehensive overview of patch attacks and defense strategies for image recognition models. It serves as a foundation for subsequent discussions and analyses regarding the application of these concepts to vision-based perception systems in autonomous driving.

### 2.1 Patch Attack Definition

The concept of adversarial patch attacks, which are generated through training, was first demonstrated by Brown et al. in 2017 [9]. These attacks represent a specific type of adversarial attack that specifically targets computer vision systems, with a particular focus on object recognition models. Unlike other adversarial attacks that manipulate individual input samples, adversarial patch attacks involve adding a visually inconspicuous patch to an object or scene. This patch is carefully crafted to exploit vulnerabilities in the object recognition model, leading to misclassification or misinterpretation of the scene by the system.

The primary objective of an adversarial patch attack is to deceive the object recognition system, causing it to misclassify or fail to recognize the objects present in the scene. The adversarial patch is strategically placed to occlude or modify specific features of the objects, prompting the system to make incorrect predictions or ignore the objects entirely. To ensure the effectiveness of the patch while remaining imperceptible to humans, various techniques are employed, including optimization algorithms and gradient-based methods.

Optimization methods originally proposed for finding adversarial examples can also be applied to generate adversarial patches by slightly modifying the loss function. One popular approach is to utilize masks to define the shape and location of the patch. By incorporating these masks into the optimization process, researchers can search for the optimal values for the patch pixels within the given constraints, resulting in visually inconspicuous yet effective patches that deceive object recognition systems.

In the broader context of finding adversarial examples, several optimization methods have been proposed. For example, the Fast Gradient Sign Method (FGSM) introduced by Goodfellow et al. [49] leverages the maximum direction of gradient change in deep neural networks to generate adversarial perturbations. This algorithm achieves high attack speed as a single-step attack but may have a relatively low success rate in generating adversarial examples. To
address this limitation, Kurakin et al. [74] proposed an iterative version of FGSM (I-FGSM) that performs multiple small steps, resulting in higher success rates in obtaining adversarial examples. [107] proposed the jacobian-based saliency map attack (JSMA), which differs from other methods by utilizing the probability information of the model’s output category instead of the gradient information of the loss function. JSMA performs back-propagation using this probability information to obtain gradient information, allowing the construction of an adversarial saliency map for the attack. JSMA quantifies the influence of each pixel on model classification using the forward gradient and constructs an adversarial significance graph based on the Jacobian matrix. By selecting the pixel positions with the highest anti-significant values, JSMA perturbs pixels to generate adversarial examples. Experiments have shown that JSMA can achieve high attack confidence rates while producing relatively small perturbations that are difficult for humans to detect. [12] introduced a CW attack that constrains $l_p$ norm, demonstrating that defensive distillation is ineffective against such attacks. The CW attack is considered one of the most powerful and versatile attack methods, as it can optimize for different norms and achieve high success rates against various defense mechanisms. [101] introduced the DeepFool algorithm, which aims to generate minimal perturbations required to misclassify an input example. The algorithm iteratively computes the minimal adversarial perturbation by linearizing the decision boundary of the deep neural network and finding the closest decision boundary intersected by the input example. DeepFool perturbs the input example along this direction until it reaches a misclassification. By minimizing the perturbation, DeepFool generates adversarial examples that are visually similar to the original inputs but result in incorrect predictions. [96] proposed the Projected Gradient Descent (PGD) attack as a powerful white-box attack against adversarial examples. PGD iteratively applies small perturbations within a specified $l_{\inf}$ norm to maximize the loss function and generate adversarial examples. It incorporates a projection step to ensure that the perturbed examples remain within an acceptable range. The PGD attack is known for its effectiveness in generating strong adversarial examples that are difficult to defend against, even with robust training methods such as adversarial training. There are also several other notable attack methods, such as the Momentum Iterative Method (MIM) [16], which combines elements of FGSM and PGD to improve attack efficiency, and the Genetic Algorithm (GA) [100], which employs evolutionary principles to search for adversarial examples. These methods highlight the diverse range of approaches that attackers can employ to exploit vulnerabilities in deep learning models. It is worth noting that as new defense mechanisms are developed, attackers continually adapt their methods to bypass these defenses. Adversarial attacks and defenses
are an ongoing research area, with new techniques being proposed to improve both attack success rates and defense robustness.

Masked Projected Gradient Descent with infinite steps (Masked PGD-inf) is an advanced adversarial attack strategy specifically designed for patch attacks. It extends the concept of Projected Gradient Descent (PGD) by incorporating a masking mechanism, which allows the attacker to focus on the most vulnerable regions of an image. More specifically, the Masked PGD-inf attack performs gradient updates in a patch region which is likely to maximize the attacker’s objective. When updating the attack at each iteration, a binary mask $M$ is used to zero-out the gradient on the rest of the image. We denote the adversarial image as $X^\delta$, where $\|X^\delta - X\| \leq \varepsilon$. Here $\varepsilon$ is a scalar parameter that controls the attack strength. The condition $\|X^\delta - X\| \leq \varepsilon$ ensures that the perturbations introduced in the adversarial image are within a specified range, allowing for a controlled level of distortion from the original image. Given an input image $X$, its ground truth label $y$, and model $f$ with weights $\theta$ and a defined loss function $L$, a masked PGD attack is generated by maximizing the loss function in an iteration way:

$$X_{t+1}^\delta[\text{patch}] = C_\varepsilon \left( X_t^\delta + \alpha \text{Sign}(\nabla_X L(X_t^\delta, y, \theta)) \right)[\text{patch}] \quad (2.1)$$

Note that the clipping function $C_\varepsilon$ is utilized to prevent the per-pixel modification from going beyond the $\varepsilon$ ball. Here, $\odot$ represents the element-wise multiplication of the mask $M$ with the gradient sign. In the context of patch attacks, the mask $M$ is used to guide the adversarial perturbation towards the most vulnerable regions of an image, which enables the attacker to deceive the classification model more effectively. By iteratively updating the adversarial image using the masked gradients, the attacker can achieve their desired objective while minimizing the overall distortion in the image. While this attack has been shown to be effective in some contexts, it frequently fails to produce strong attacks against detection based defenses. We next introduce a modified objective which is specifically tailored to attack detection-based approaches.

Adversarial patch attacks pose significant challenges to the robustness and reliability of computer vision systems, particularly in real-world scenarios where autonomous vehicles or surveillance systems heavily rely on accurate object recognition. These attacks have raised concerns about the security and trustworthiness of such systems, as they can be
potentially exploited to deceive or manipulate the perception module of autonomous vehicles or surveillance systems.

To mitigate adversarial patch attacks, researchers and practitioners are actively exploring defense strategies and countermeasures. These may include adversarial training, where the system is exposed to adversarial examples during the training phase to enhance its robustness against such attacks. Other approaches involve using anomaly detection mechanisms, incorporating defensive distillation techniques, or exploring the integration of multiple detection models to improve resilience against adversarial patch attacks.

2.2 Patch Attacks for Image Classification and Object Detection

**Patch Attacks for Image Classification** In 2017, [9] firstly proposed a universal and targeted attack on real-world physical objects for attacking image classification problem. Although primitive, this attack was also highly robust and practical, building a foundation for subsequent adversarial patches. Brown et al. [9] also presented camouflaged patches under constraints to force the similarity between the final patch and the starting patch. This attack is usually created in a relatively larger size and consequently is more evident to human eyes. [9] claimed that the region with the patch posing a security concern would become the most salient feature in the image, which was invalidated and thus distinguished it from LaVAN [65]. Karmon et al. [65] proposed LaVAN (Localised and Visible Adversarial Noise) in a smaller size than the adversarial patches proposed by [9] with similar attack effectiveness, focusing more on the model weaknesses that led to misclassification. It was the first to introduce some stealth properties in patch attacks. [65] conducted patch perturbations in both images and networks for which the latter was more effective though not physically attainable. While it was more effective, it lacked robustness across transformations (like rotation) and locations over the image. Another attempt to make the adversarial patch less suspicious to human eyes was the adversarial QR patch [20], [21]. It was created using a masked patch initialized with a QR pattern and trained later to make successful attacks. QR patches opened the door to a new dimension where adversarial patches were trained using QR codes or some other strict patterns to mislead the human intuition. However, the harder one tried to obfuscate the patch from human detection, the less effective these attacks were. As we can see from
the three aforementioned concepts, patch attacks are effective yet noticeable due to their irregular structure and unnatural appearance in the scene. Most efforts that are taken to reduce the identification of patches are observed to lead to the sacrifice of attacking behaviors. The reasons behind this stem from the fact that producing stronger attacks requires large perturbations without many structural constraints like in the QR patch \[20], \[21] and that the lack of constraints on perturbations leads to irregular and random patterns in the perturbation learning. Hence there exists a trade-off between patch identification and attack effectiveness. However, \[87] argued that understanding the network perceptual sensitivity to adversarial patches could help to design more visually natural patches with strong attacking capability, which had been overlooked by existing patch attacks by then. They designed PS-GAN (Perceptual-Sensitive Generative Adversarial Networks) to improve the visual fidelity and enhance the attacking ability, which was also demonstrated to have good transferability across network structures. Alternatively, \[48] proposed an image reconstruction technique using deep image prior (DIP) \[143] to develop imperceptible perturbations that were robust to affine deformations, namely an attack based on local patches. They also claimed that the proposed image reconstruction helped achieve greater flexibility for perturbations across the whole image. \[173] proposed DiAP (Data-independent Adversarial Patch) to fool the target model without any knowledge of the training data. In this technique, non-targeted attacks were generated by optimizing a spurious activation objective to deceive the features learned on each layer in the model and then were transformed into targeted ones by extracting important features from the background of the target class.

**Patch Attacks for Object Detection**  
The transferability of adversarial patches trained for classification often fails to apply to object detection due to inherent differences in their targets \[89]. Object detection models are designed to locate and classify objects within proposed regions, whereas classification tasks focus on correctly classifying a single object in an image. Adversarial patches for classification tasks mislead CNN models by generating more salient features than those present in the image \[9]. However, in object detection tasks such as \[118], \[46], \[45], \[120], multiple region proposals are generated to locate and classify objects. Consequently, attacking object detection requires targeting several proposed objects and their bounding boxes instead of a single salient feature. \[89] extended the adversarial attacks to object detection with DPatch (adversarial patch attack on object detectors). DPatch concurrently attacks bounding box regression and object classification by iteratively training the patch. DPatch achieves both untargeted and targeted attacks, demonstrating
decent attack transferability across datasets. However, DPatch is limited to digital image scenarios. To address this limitation, [79] proposed modifications to enhance the power of patch attacks and extend their applicability to real-world object detection. [96] introduced the PGD (Projected Gradient Descent) technique, which enforces the values of pixels to remain within the permitted boundary, allowing for transferability of attacks to the real world. However, the effectiveness of these patches decreases as the distance between the patch and the objects increases. Increasing the patch size can lead to a stronger adversary, as observed in the work of [152] with DPAttack (Diffused Patch Attacks), where a small area of the image is perturbed to effectively attack multiple features in the scene. [60] further refined this approach with RPAttack (Refined Patch Attack), perturbing fewer pixels to create imperceptible attacks while retaining effectiveness. They used the knowledge of key pixels to refine the patch and removed pixels that had less impact on the attack, reducing unnecessary perturbations. RPAttack also employed ensemble learning during the training phase to ensure patch robustness across different model architectures. [138] attempted to fool models in surveillance cameras designed to detect trespassing into restricted areas. This posed a greater challenge because the attack needed to be equally effective against people with different colors, sizes, clothing, orientations, and poses. People exhibit more variations in terms of shapes and appearances compared to road signs, which are usually consistent. [138] proposed a loss objective to reduce the classification score, resulting in attacks that performed exceptionally well in practice. However, these attacks required strong conditions on the location and lacked transferability across model architectures. Another application of fooling cameras was the camouflage of military assets against aerial detection, an extension of [138] proposed by [29]. They slightly modified the loss function to make the patch difficult for human eyes to detect. [29] demonstrated a trade-off between patch size and performance, showing that larger patches placed exactly over the asset of interest were most effective, but smaller patches, including less colorful ones, yielded better performance. This highlighted the importance of patch location in the attack. As illegal drone usage increased for surveillance near military or defended areas, patch camouflage applications became crucial and potentially viable threats. Additionally, [93] proposed Patch-Noobj, which adaptively scales the patch size based on the size of the attacked aircraft, demonstrating attack transferability across models and datasets. Static patches that remain fixed in the scene [89],[38], [138],[29], [79] become less effective when the camera’s relative position changes with respect to the attacked image, such as a moving car on the road. The patch’s performance becomes uncertain due to changes in camera angle and field of view, leading to different sizes of target objects in each frame. Most
of these patches work well for planar objects but struggle with non-planar objects commonly encountered in real-world scenarios. To address this, [57] designed the Dynamic Adversarial Patch, which remains invariant to the camera’s position by switching between trained patches, making the attack dynamic for the upcoming scene. Multiple screens were placed at different locations to attack detection when the camera’s viewpoint changed or multiple cameras were present. Semantic adversary features were introduced to prevent semantically related classes from having the same influence in autonomous driving scenarios. The dynamic patch was the first attempt to make an adversary adaptable to dissimilar situations, but future research is needed to explore its transferability across architectures, environments, models, and the cost requirements associated with screens or LEDs. In addition to attacking objects, [176] proposed the translucent patch to be applied on camera lenses. While most attacks focus on attacking objects, this patch selectively attacks only the target object, leaving the rest of the scene untouched. Crafting such a patch is more challenging than attacking objects in general, as demonstrated by the necessary considerations, including patch structure, region-level patch blending, shape positioning, and shearing. [176] empirically showed that the attack was transferable across model architectures like R-CNN. [147] introduced the invisibility patch, which aims to make the target object invisible to the detector, attacking target classes only in the scene. The patch is trained iteratively to minimize the loss of the detection score specifically for the target class. This attack demonstrates high transferability across datasets, architectures, and from the digital to the physical world. Instead of using posters or stickers, [147] suggested displaying patches on portable screens, which showed good performance in the physical world. However, this attack requires the patch to be precisely positioned over the target image, and the need for a hardware screen increases the cost of performing the attack. [75] proposed AGAP (Attention-Guided digital Adversarial Patch), which differs from most existing attacks using random gradient descent. AGAP utilizes high feature density regions in the image to calculate the location and size of the generated patch, enhancing its effectiveness.

### 2.3 Adversarial Patch Detection and Defense

We begin by introducing a saliency-based defense method that aims to detect adversarial patches. Firstly, [54] proposed DW (Digital Watermarking) defense. They constructed a saliency map of the image to assist in patch removal and adversarial image masking, effectively
blocking adversarial perturbations. However, this empirical defense lacked guarantees against adaptive adversaries, serving as a foundational concept for subsequent certified defenses against patch attacks. [103] introduced LGS (Local Gradient Smoothing), which aimed to suppress highly activated and perturbed regions in the image without impacting salient objects. By regularizing irregular gradients in the image before passing them through a deep neural network (DNN) model, LGS achieved robustness with minimal clean accuracy degradation. Unlike its counterparts that processed the entire image globally, LGS focused on local region processing. [22] proposed SentiNet, the first architecture for localized universal attacks, which leveraged the unique behavior of adversarial misclassification to detect attacks without prior knowledge of trained models or adversarial patches. Salient regions were used to observe the model’s behavior, and SentiNet demonstrated empirical robustness and effectiveness even in real-world scenarios. However, the evaluation of adversarial regions through subtraction of the suspicious region occasionally resulted in false adversarial region proposals. Moreover, the random placement of suspicious adversarial regions in the preserving image could occlude main objects in the scene, leading to incorrect predictions. In response, [17] proposed, a defense to detect and mitigate robust and universal adversarial patch attacks by leveraging the localized nature of the attacks via image inpainting. They used a modified saliency map [135] to detect highly active perturbed regions and strategically placed suspicious extracted regions in the least salient areas of the preserved image to avoid occlusion with main objects. Jujutsu outperformed other empirical defenses in terms of robust accuracy and low false-positive rate (FPR) across datasets, patches of various shapes, and attacks targeting different classes.

The second research line in defense against adversarial examples involves adopting adversarial training [48]. VaN (Vax-a-Net) [48] was the first training-time defense proposed against patch attacks. Two defenses, namely DW (Digital Watermarking) [54] and LGS (Local Gradient Smoothing) [103], exploited highly active visual behaviors on saliency maps during inference. VaN utilized a modified version of DC-GAN (Deep Convolutional Generative Adversarial Network) [114] to synthesize effective adversarial patches and simultaneously train the model to defend against those patches. Additionally, it improved LaVAN [65] by incorporating location optimization, which increased computational cost due to the exploration of all possible locations [48]. PG (PatchGuard) [158] presented a defensive technique against adversarial patches using CNNs with small receptive fields to build robust classifiers. The small receptive field limited the number of features influenced by the attack and facilitated the identification of feature boundaries. PG employed a feature aggregation method to mask and recover
correct predictions, aiming to maintain high clean accuracy against localized adversarial patches. PG++ (PatchGuard++) [157] further improved upon PG by introducing feature extraction for patch attack detection. It demonstrated significant enhancements in both provable robust accuracy and clean accuracy. DG (DetectorGuard) [156] achieved provable robustness against hidden localized patches, providing formal guarantees in an adversarial setting. DG focused on securing object detectors primarily in domains such as autonomous driving, video surveillance, and identity verification, which differed from most adversarial defenses concentrated on image classification [156]. Many defenses proposed in the context of adversarial patches rely on pre-processing of inputs during inference [54, 103]. However, Chiang et al. [19] argued that these defenses are vulnerable to white-box adversaries, leading to the development of certified defenses. Certified defenses not only protect against patch attacks but also provide guaranteed confidence in their defensive capabilities [19].

Finally, we introduce the certified defenses, which involve evaluating extreme bounds to certify models under worst-case scenarios, are generally computationally expensive compared to their counterparts. Despite the time cost limitations, certified defenses represent an essential step towards achieving ultimate robustness in deep learning-based vision systems. One proposed certified defense against patch attacks is CertIBP (IBP certified models) [19], which introduced faster training methods and shared similarities with IBP (Interval Bound Propagation) [50] and CROWN-IBP [168]. CertIBP demonstrated superior certified accuracy compared to the empirical accuracy of LGS [103] and DW [54], and it exhibited robustness against patch attacks of various shapes. However, its scalability was demanding due to the quadratic increase in computational burden with image size. Levine et al. [80] utilized randomized smoothing, commonly used in certified defenses against attacks such as [77] and [81], and extended its robustness to DRS (De-Randomized Smoothing) for certified defenses against patch attacks. By exploiting the more constrained setting of patch attacks compared to $l_0$ attacks [81], they employed a de-randomized procedure that incorporated knowledge of the patch structure, providing guaranteed robustness against generic patch attacks. Another certified defense technique, RCD (Randomized Cropping Defense), was proposed by [85]. RCD classified randomly cropped subsets of the original image independently and classified the original image based on the majority vote of the predicted classes of the sub-images. It achieved comparable clean accuracy, faster inference time, and higher certified accuracy under worst-case scenarios compared to DRS [80] and PG (PatchGuard) [158]. Certified defenses ideally include certification as part of their training objectives to avoid post-hoc calibration. BagCert [99], inspired by [8], combined a specific model architecture with a certified training
procedure, enabling scalability to larger patches and providing higher accuracy compared to CertIBP [19]. BagCert achieved good accuracy even with small receptive fields, reducing the regions affected by the adversarial patch in the final feature map. It outperformed CertIBP in terms of certified accuracy, scalability, and certification time on CIFAR10 and ImageNet datasets. Additionally, other certified defenses such as DRS [80], PG [158], and CROWN-IBP [168] involved inference time computations for certified robustness instead of actual model training. [97] employed occlusions during adversarial training to detect and defend against adversarial patches at inference time. It demonstrated better clean and certified accuracy and improved effectiveness against adaptive attacks. However, it made assumptions that might be infeasible in practical scenarios, such as knowing the size of patch attacks during training. Arvinte et al. [2] introduced a two-stage detection process using image residuals for patch-based attacks, claiming better generalization and effectiveness in detecting patch adversarial attacks. Co et al. [24] proposed HyperNeuron, enabling real-time detection of UAPs (Universal Adversarial Perturbations) [100] by identifying suspicious neuron hyper-activations. It provided defense against adversarial masks and patch attacks with lower latency, making it suitable for real-time applications. These various certified defense techniques contribute to the development of robust defenses against patch attacks, offering different trade-offs in terms of accuracy, scalability, and computational cost.
In this chapter, we delve into the related work on the challenges posed by adversarial attacks and the defense mechanisms developed to mitigate them in the context of autonomous driving.

Our discussion on adversarial attacks is divided into two main parts: attacks on the perception module and attacks on the localization module. Firstly, we focus on attacks targeting the perception module of autonomous vehicles. The perception module plays a critical role in understanding the surrounding environment through various sensors such as cameras, Lidar, and radar. Adversarial attacks on the perception module aim to manipulate sensor inputs to deceive the perception algorithms, leading to misclassification or incorrect object detection. We examine notable research studies that explore different attack strategies, including remote attacks on camera and Lidar sensors, contactless attacks against sensors, and vulnerability assessment of ultrasonic sensors. Secondly, we shift our attention to attacks on the localization module of autonomous driving systems. The localization module is responsible for accurately determining the vehicle’s position and orientation within the environment. Adversarial attacks on the localization module can disrupt the accurate estimation of the vehicle’s location, leading to potential navigation errors. We review research efforts that investigate the vulnerability of the localization module to adversarial attacks and discuss the impact of such attacks on the overall autonomous driving system.

Lastly, we introduce the papers on developing defense mechanisms. These mechanisms aim to enhance the robustness of autonomous driving systems against potential attacks and ensure
their reliable performance in real-world scenarios. We discuss various defense strategies, including adversarial training, ensemble methods, distillation techniques, certified robustness, and input transformation approaches.

3.1 Adversarial Manipulation of Sensor Inputs in Autonomous Driving: Attacking the Perception Module

We start with introducing attacks for traffic signs, which explores the specific adversarial attacks aimed at manipulating the recognition and interpretation of traffic signs by autonomous vehicles. These attacks can involve altering the appearance of traffic signs through carefully crafted modifications, such as adding subtle perturbations or occlusions. By exploiting the vulnerabilities of the perception algorithms responsible for traffic sign detection and classification, adversaries can deceive the autonomous vehicle into misinterpreting or disregarding important traffic sign information.

The recognition of traffic signs is a commonly employed application of deep neural network (DNN) models, finding utility in diverse scenarios like automatic driving for road sign recognition. Consequently, a series of studies have emerged to investigate the security vulnerabilities associated with these models. [38],[134] first demonstrated that the road sign recognition model is vulnerable to physical adversarial attack. The author proposed to optimize multiple white-black blocks with rectangle shapes. Specifically, the author first collected considerable physical traffic sign images as the training set to improve the robustness of adversarial patches to various physical environmental conditions. Another line of research is about crafting generative attack, [87] proposed to utilize the generative model to construct inconspicuous adversarial patches. Specifically, the author exploited the attention map extracted by the Grad-CAM [127] to guide the paste location of adversarial patches for better attack performance. They adopt GAN loss, adversarial loss, and the $l_2$ norm loss that is calculated by the seed patch and the output of the generator to optimize the perturbation. [36] proposed a novel approach to camouflage the adversarial perturbation (i.e., AdvCam) into a natural style that appears legitimate to human observers. Specifically, they exploited the style transfer technique [43] to hide the large magnitude perturbations into customized styles, which makes attacks more stealthy. [69] proposed a novel method to learn physical transformation from videos recorded in the real world by using a generative model (i.e.,
PhysGAN), which is then used to optimize the physical robust adversarial patch. Specifically, the author placed the adversarial poster on the target object of the video frame and fed it into a video feature extractor. Recently, [172] proposed to utilize the natural phenomenon (i.e., shadow) to perform physical attacks. Additionally, the license plate recognition (LPR) model has been demonstrated to be vulnerable to physical adversarial attacks [113]. Rather than optimize the perturbation over the while image pixel, [113] proposed to optimize the best position where to paste the adversarial patch.

Besides traffic signs, we can put adversarial patch to other places for manipulating self driving cars or altering the perception of other objects and entities in the environment. Examples include modifying the appearance of vehicles or pedestrians to confuse the perception algorithms, attaching screens or projections displaying adversarial patterns on surrounding objects, or even deploying physical objects strategically to generate false sensor readings. These attacks exploit the weaknesses in the perception algorithms’ ability to accurately identify and classify objects in the environment, potentially leading to hazardous or incorrect decision-making by the autonomous vehicle. [165] introduced a method for generating adversarial license plates to deceive an object detection model. In a similar vein, [57] proposed a dynamic adversarial patch that utilizes a digital screen to evade object detection models. More recently, [128] developed a targeted attack approach to create adversarial patches that are applied to the car hood, aiming to attack object detectors. [153] proposed a searching algorithm for finding adversarial texture of vehicles. These adversarial pattern will decrease the chance where the adversarial will be detected by a victim autonomous driving model.

There are various attacks aimed at degrading the quality of sensor data by introducing noise into the environment. For instance, [111] propose techniques to blind cameras by emitting intense light into the autonomous vehicle’s camera. [161] experiment with a blinding attack using lasers to cause temperature damage to cameras. [131] propose a blinding attack for Lidar, where the Lidar is exposed to a strong light source of the same wavelength, causing it to fail in perceiving objects in the direction of the light source. Jamming attacks on ultrasonic sensors and radars are investigated in [161], where an ultrasound jammer is used to attack the parking assistance system of multiple vehicles, rendering them unable to detect surrounding obstacles. To attack radar systems, electromagnetic waves generated by a signal generator and frequency multiplier are directed at the Tesla Autopilot system, compromising its functionality. Similarly, [84] simulates a jamming attack on ultrasonic
sensors, demonstrating that strategically placed ultrasonic sensors can significantly interfere with the readings of the targeted sensor.

3.2 Adversarial Attacks on Localization in Autonomous Driving

In the context of self-driving cars, the localization system plays a critical role in accurately determining the vehicle’s position and aligning it with the collected sensor data and HD maps. One common approach for localization involves using GPS as reference and comparing the point cloud data obtained from lidar and radar sensors with the high-definition (HD) maps stored in the cloud.

Cloud Attack  Cloud attacks pose a significant threat to the localization system by targeting the communication between the self-driving car and the cloud or centralized server responsible for information sharing. Adversaries may exploit vulnerabilities in the communication channel to disrupt the flow of data or manipulate the information exchanged between the vehicle and the cloud. By compromising the cloud, adversaries can tamper with or block the transmission of HD maps to the vehicle. As a result, the self-driving car may not have access to the most up-to-date and accurate map information for comparison with its estimated location. This can lead to incorrect localization, misalignment with the surrounding environment, and potentially unsafe navigation. Disrupting the flow of data between the vehicle and the cloud can also impact real-time localization updates. Without continuous access to the cloud’s resources, the vehicle’s localization system may suffer from delayed or outdated information, hindering its ability to accurately determine its position and make informed driving decisions. Therefore cloud is often a prime target for adversaries due to the continuous communication between the cloud and autonomous vehicles, leading to potential instability in the vehicles. The updating of HD Maps in real-time via Vehicle-to-Everything (V2X) communication can be susceptible to control by attackers. For example, Sybil attacks and message falsification attacks, as described in [133], aim to disrupt the automatic navigation system’s efficiency. Sybil attacks involve creating numerous "fake drivers" with fabricated GPS information in the target location system, deceiving the system and impacting localization and navigation tasks by causing traffic congestion. Message falsification attacks involve intercepting and
tampering with traffic information exchanged between vehicles and the HD map server, spoofing other vehicles during the HD map update process. Traditional cloud attacks pose threats to the V2X network, where autonomous vehicles exchange information. Denial of Service (DoS) and Distributed DoS (DDoS), as outlined in [92], [35], can exhaust service resources, resulting in high latency or even network unavailability in the V2X network. In such cases, autonomous vehicles may struggle to connect to the HD map for accurate navigation and perception services, significantly jeopardizing their safety. Another variant of attack targets the over-the-air (OTA) channel in the cloud. Attackers can hijack the data transfer channel between the cloud and an autonomous vehicle, injecting malware into the vehicle [106].

**Spoofing Attack** The consequences of spoofing attacks can be severe, as the perception and decision-making systems of self-driving cars heavily rely on accurate localization information. If the vehicle’s localization system is compromised, it may incorrectly perceive its surroundings, leading to incorrect object detection, path planning, and control decisions. This can result in hazardous situations, including collisions, incorrect lane changes, or other dangerous maneuvers. In [111], a spoofing attack on Lidar was conducted. By delaying the real output signal and creating a counterfeit signal, the distance calculation between the vehicle and objects was manipulated, causing Lidar to detect objects at incorrect distances. Similarly, spoofing attacks were implemented against ultrasound sensors and radar [109],[162].

GPS systems are also vulnerable to spoofing attacks. In GPS spoofing attacks, adversaries emit false or noise signals that mimic legitimate GPS signals. These spoofed signals are designed to deceive the GPS receiver in the self-driving car, leading to inaccurate position estimations. By manipulating the GPS signals received by the vehicle, attackers can intentionally distort the localization process and mislead the car about its actual position. GPS spoofer can deviate a yacht from its intended route [112]. GPS spoofing devices have been developed to block legitimate signals and manipulate navigation systems, causing vehicles to deviate from their original routes [98],[148],[167]. Cameras can also be targeted by spoofing attacks. Altering the appearance of the ground plane captured by optical-flow cameras can manipulate the processing of optical-flow information, enabling control over unmanned aerial vehicles [28]. Another type of spoofing attack is relaying, where the original signal is re-sent from a different position. LiDAR relaying attacks resulted in the detection of ghost walls in different locations [111]. In [104], a projector was used to display spoofed traffic signs on vehicle
cameras, leading to the misinterpretation of fake signs as real ones. We will defer the extensive literature review on GPS spoofing attack to Chapter 6 in the related work section.

3.3 Defense Mechanisms for Enhancing Robustness of Autonomous Driving System against Malicious Attacks

Among all the countermeasures for physical sensor attacks, redundancy [111], [162], [84] is the most promising strategy to defend jamming attacks. Redundancy involves deploying multiple sensors to collect data of the same type and fusing them for perception. For example, when attackers commit the blinding attack on one camera, others could still collect normal images for environment perception. However, redundancy leads to increased financial costs and sensor data fusion is considered an intractable research issue. Other approaches to enhance camera robustness include using near-infrared-cut filters during the day [111] and photochromic lenses to filter specific types of light [111]. Detection systems can be built for ultrasonic sensors, radars, and GPS to detect and mitigate jamming attacks [162].

To defend against spoofing attacks, introducing randomness into data collection is effective [111], [162]. Attackers find it harder to send fake signals when the probe time is set randomly for Lidar. Data fusion from cameras, Lidar, radars, and ultrasonic sensors can help stabilize the performance of the perception layer.

Various defense methods have been proposed for adversarial attacks. Proactive defense methods include adversarial training, network distillation, network regularization, model ensemble, and certified defense [49], [72], [140], [108], [73], [88], [77],[115], [151]. Reactive defense methods ([171], [78], [160], [52], [125], [64], [83]) include adversarial detection and adversarial transformation to detect and counter adversarial examples.
Chapter 4

Dataset and Simulation Environment for Image Recognition and Autonomous Driving Research

In this chapter, we present the experiments conducted as part of the research for this PhD, along with the datasets utilized for training and evaluation purposes. The experiments aimed to assess the performance and robustness of the proposed methods in the context of self-driving car localization and perception.

**Carla Simulator** To create a realistic virtual environment for the experiments, we employed the Carla Simulator. Carla \[11\] is an open-source simulator designed for autonomous driving research. It provides a high-fidelity simulation platform that offers various sensor modalities, including cameras, lidar, and radar, as well as a dynamic and customizable urban environment. The Carla Simulator enabled us to generate synthetic data and simulate real-world driving scenarios, allowing for controlled experimentation and evaluation of the proposed algorithms.

**Uber Movement** Uber Movement \[142\] offers aggregated and anonymized data on travel times and traffic patterns derived from the millions of Uber trips that occur in cities around the world. This data can be useful for urban planning, transportation analysis, and research purposes. With Uber Movement, users can access historical and real-time traffic information to understand traffic conditions and make informed decisions about routes and travel times.

**OpenStreetView** OpenStreetView (OSV) \[105\] was another valuable resource used in this research. OSV is an open-source project that provides a vast collection of street-level
imagery from locations around the world. These images are captured by volunteers using GPS-enabled devices, making them a valuable resource for studying real-world environments.

It offered several advantages for our research. Firstly, it provided a unique perspective of street-level scenes, allowing us to analyze the visual content and characteristics of real-world road environments. This was particularly relevant for the evaluation of algorithms related to autonomous driving and scene understanding.

**Computer Vision Datasets** To train and evaluate the performance of the computer vision models used in the self-driving car perception tasks, we utilized several popular computer vision datasets. These datasets encompassed a wide range of visual recognition tasks, enabling comprehensive analysis and benchmarking of the proposed methods. The following datasets were employed:

- **IMAGENETTE and IMAGENET**: IMAGENETTE and IMAGENET are subsets of the large-scale ImageNet dataset [30]. IMAGENETTE consists of 10 easily classified classes, while IMAGENET contains the full ImageNet dataset with over 1.2 million images across 1,000 categories. These datasets facilitated training and evaluating deep convolutional neural networks (CNNs) for image classification tasks.

- **MNIST**: The MNIST dataset [76] is a widely-used benchmark for handwritten digit recognition. It contains 60,000 training images and 10,000 test images of handwritten digits (0-9), making it suitable for evaluating algorithms in the context of digit recognition and classification.

- **CIFAR-10 and CIFAR-100**: The CIFAR-10 and CIFAR-100 datasets [71] consist of 50,000 training images and 10,000 test images, divided into 10 and 100 classes, respectively. These datasets are commonly used for evaluating image classification models, particularly CNNs, on more complex and diverse visual recognition tasks.

- **VGGFace**: The VGGFace dataset [110] is a comprehensive dataset for face recognition. It contains over 2.6 million images of 2,622 celebrities, allowing for the development and evaluation of face recognition models in real-world scenarios.
• **CelebA**: CelebA [91] is a large-scale face attribute dataset that consists of over 200,000 celebrity images. It includes annotations for various facial attributes, making it suitable for training models for facial attribute prediction and analysis tasks.

• **Flickr**: The Flickr dataset [42] is a collection of images gathered from the popular photo-sharing platform Flickr. It contains a diverse set of images covering various scenes, objects, and visual concepts, providing a rich dataset for training and evaluating models in the context of image recognition and understanding.

By utilizing these diverse datasets, we were able to thoroughly assess the performance and generalization capabilities of the developed algorithms across different visual recognition challenges. The use of these datasets ensured that the proposed methods were rigorously evaluated and validated, providing a solid foundation for the conclusions and findings of this PhD research.
Part II

The Vulnerabilities of Autonomous Driving System and Connected Autonomous Driving Fleet
Chapter 5

A Fast and Differentiable Adversarial Testing Framework for Simulated Autonomous Driving

5.1 Introduction

Computer vision has made revolutionary advances in recent years by leveraging a combination of deep neural network architectures with abundant high-quality perceptual data. One of the transformative applications of computational perception is autonomous driving, with autonomous cars and trucks already being evaluated for use in geofenced settings, and partial autonomy, such as highway assistance, leveraging state-of-the-art perception embedded in vehicles available to consumers. However, a history of tragic crashes involving autonomous driving, most notably Tesla [137] and Uber [53] reveals that modern perceptual architectures still have some limitations even in non-adversarial driving environments. Even more concerning is the increasing abundance of evidence that state-of-the-art deep neural networks used in perception tasks are vulnerable to adversarial perturbations, which involve either imperceptible noise added to an input image and designed to cause misclassification [49, 166] or unsuspicious perturbations that modify scenes, rather than merely input images [74, 39, 134, 37].

One of the central challenges in ensuring that autonomous driving architectures are robust is that the scale of necessary testing far exceeds what is feasible in a physical environment. Consequently, simulation-based testing is a critical element of autonomous driving platform development and evaluation [33, 149]. On the other hand, studies of vulnerabilities in deep neural network architectures that are used in autonomous driving predominantly evaluate perception in isolation from control. However, in simulation-based adversarial testing, it
is crucial to consider the interplay between perception and control for two reasons. First, autonomous driving is a dynamical system, so that a fixed adversarial perturbation to a scene is perceived through a series of highly interdependent perspectives. Second, even if we succeed in causing the control outputs of self-driving cars to deviate from normal, the vehicle will now perceive a sequence of frames that is different from those encountered on its normal path, and typically deploy self-correcting behavior in response. For example, if the vehicle is driving straight and then begins swerving towards the opposite lane, its own perception will inform the control that it’s going in the wrong direction, and the controller will steer it back on course. These two considerations are largely absent from prior approaches for generating adversarial perturbations, making these inappropriate for adversarial testing of full-stack autonomous driving in simulation experiments.

To address these limitations, [7] recently proposed Bayesian Optimization (BO) as a way to design simple physically realizable (that is, easy to implement in a physical scene) adversarial examples for adversarial testing of end-to-end autonomous driving architectures in Carla autonomous driving simulations [33]. The key challenge with this approach, however, is that attack design must execute actual simulation experiments for a large number of iterations (1000 in the work above), making it impractical for large-scale simulation-based adversarial testing.

Figure 5.1: Overview. We collect and calibrate frames from the unmodified environment (shown in the green box), and given a choice of attack pattern parameters, composite the pattern to create approximate renderings of frames corresponding to placing the pattern in the environment. Our composition function is differentiable with respect to the attack pattern parameters, and we are thus able to use end-to-end gradient-based optimization when attacking a differentiable control network, to cause the network to output incorrect controls that cause the vehicle to deviate from its intended trajectory (from the green to the blue trajectory, as shown in the right column), and crash.
We propose a highly scalable framework for designing physically realizable adversarial examples for adversarial testing of simulated end-to-end autonomous driving architectures. Our framework develops a differentiable pipeline for digitally approximating driving scenarios, and is illustrated in Figure 5.1. The proposed approximation makes use of image compositing, learning homography and color mappings from a birds-eye view of embedded adversarial examples to projections of these in images based on actual driving frames, and sampling sequences of actual frames with small perturbations to control to ensure adequate sampling of possible perspectives. The entire process can then be fed into automatic differentiators to obtain adversarial examples that maximize a car’s deviation from its normal sequence of controls (e.g., steering angle) for a target driving scenario.

We then evaluate the proposed framework using Carla simulations in comparison with the state-of-the-art BO method, with adversarial perturbations generalizing rectangular occlusions drawn on road pavement proposed by [7]. Our experiments show that the resulting attacks are significantly stronger, with effects on induced deviations and road infractions often considerably outperforming BO, using an order of magnitude fewer simulation runs. Furthermore, we show that our approach yields adversarial test instances that are robust to variations in weather and visibility.

5.2 Proposed Method

Autonomous driving systems are equipped with decision algorithms that produce control signals for a vehicle based on high-level instructions—such as a given route or destination—and inputs from cameras and other sensors that make continuous measurements of the vehicle’s physical environment. We assume that the decision algorithm is in the form of a differentiable function—such as a neural network—that maps video frames from the camera, along with other inputs, to the control outputs. Given such a network or function, our goal is to determine if it is vulnerable to attack. Specifically, we seek to build a scalable and efficient method to find modifications that can be applied to a simulated autonomous driving environment with a sophisticated physics engine, and result in a stream of video frames which cause the control network to produce output signals that disrupt the vehicle’s operation, moving it away from the expected ideal trajectory.
This task is challenging since the relationship between modifications to the simulated physical environment and the network’s inputs is complex: the video frames correspond to images of the environment from a sequence of changing viewpoints, where the sequence itself depends on the network’s control outputs. The precise effect of any given modification can be determined only by actually driving the vehicle in the modified simulated environment that uses a physics engine. However, it is expensive to use such a simulator when searching for the right modification, since the process is not differentiable with respect to parameters of the modification, and would require repeated trials with candidate modifications in every step of the search process.

Instead, we propose a fast approximation to produce video frames for a given environment given a candidate modification that is differentiable with respect to parameters of the modification. Our approach requires a small number of initial simulated calibration runs, after which the search for optimal parameters can be carried out with end-to-end gradient-based optimization. Specifically, we consider the case when the modification takes the form of figures (such as rectangles) drawn on a restricted stretch of the road, and task the optimization with finding their optimal shape and color so as to maximize deviation from the controller’s trajectory prior to modification. We now describe our model for the physical modification, our approximate mapping to create video frames for a given modification, and our optimization approach based on this mapping.

### 5.2.1 Parameterized Scene Modifications

We assume modifications are in the form of a collection of $K$ figures (e.g., rectangles) that will be painted on a flat surface in the environment (e.g., road). Let $\Phi = \{x^S_k, x^C_k\}_{k=1}^{K}$ denote the parameters of this modification, with $x^S_k$ corresponding to the shape parameters, and $x^C_k$ the RGB color, of the $k^{th}$ figure. These parameters are defined with respect to co-ordinates in some canonical—say, top-down—view of the surface.

We let $M(n_c; x^S)$ denote a scalar-valued mask that represents whether a pixel at spatial location $n_c \in Z^2$ in the canonical view is within a figure with shape parameters $x^S$. This function depends simply on the chosen geometry of the figures, and has value of 1 for pixels within the figure, 0 for those outside, and real values between 0 and 1 for those near the boundary (representing partial occupancy on a discrete pixel grid to prevent aliasing artifacts).
Since the spatial extents for different figures may overlap, we next account for occlusions by assuming that the lines will be painted in order. Accordingly, we define a series of visibility functions \( V_k(n_c; \Phi) \), each representing the visibility of the \( k^{th} \) figure at pixel \( n_c \), after accounting for occlusions. We set the function for the last figure as \( V_K(n_c; \Phi) = M(n_c; x^K_S) \), and for the other figures with \( k < K \),

\[
V_k(n_c; \Phi) = M(n_c; x^S_k) \prod_{k'=k+1}^{K} (1 - V_{k'}(n_c; \Phi)) .
\]

(5.1)

5.2.2 Approximate Frames via Compositing

The next step in our pipeline deals with generating the video inputs that the controller network is expected to receive from a modified environment for given parameter values \( \Phi \). These frames will represent views of the environment, including the surface with the painted figures, from a sequence of viewpoints as the car drives through the scene. Of course, the precise viewpoint sequence will depend on the trajectory of the car, which will depend on the control outputs from the network, that in turn depends on the frames. Instead of modeling the precise trajectory for every modification, we consider a small set of \( T \) representative trajectories, corresponding to those that the vehicle will follow when driven with small perturbations around control outputs, when operating in the unmodified environment. One trajectory involves driving the car with the actual output control signals. To generate others, we consider two variants: 1) adding random noise to control outputs (Random), and 2) adding trajectories in pairs, one with random deviations to the left only, and the other only including random deviations to the right (termed Group). Given the fact that actual control is closed-loop, it is not evident that either variant of this simple approach would work; however, our experiments below (using \( T = 3 \)) show that both ideas are remarkably effective. This gives \( T \) sequences of video frames, one for each trajectory, where we assume each sequence contains \( F \) frames. We let \( \tilde{I}_f^t(n) \) denote the \( f^{th} \) “clean” image in the \( t^{th} \) sequence, representing a view of the environment without any modifications. Here, \( n \in \mathbb{Z}^2 \) indexes pixel location within each image, and the intensity vector \( \tilde{I}_f^t(n) \in \mathbb{R}^3 \) at each location corresponding to the recorded RGB values. These clean images can be obtained by driving the car—actually, or in simulation—in the original environment.
For each frame in each sequence, we also determine a spatial mapping \( n_c = G_{tf}(n) \) that maps pixel locations in the image to the canonical view. We model each \( G_{tf}(n) \) as a homography: the parameters of which can be determined by either using correspondences between each image and the canonical view of the surface—from calibration patterns rendered using the simulator, or from user input—or by calibrating the vehicle’s camera and making accurate measurements of ego-motion when the vehicle is being driven. Additionally, we also determine color mapping parameters \( C_{tf} \in \mathbb{R}^{3 \times 3}, b_{tf} \in \mathbb{R}^3 \) for each frame representing an approximate linear relationship between the RGB colors \( x^C \) of the painted figures, and their colors as visible in each frame. These parameters are also determined through calibration. Given this set of clean frames and the geometric and color mapping parameters, we generate corresponding frames with views of the modified environment simply as:

\[
I_{tf}^t(n; \Phi) = \left( 1 - \sum_{k=1}^{K} V_k(G_{tf}^t(n); \Phi) \right) \tilde{I}_{tf}^t(n) + \sum_{k=1}^{K} V_k(G_{tf}^t(n); \Phi) \left( C_{tf} x^C_k + b_{tf} \right). \tag{5.2}
\]

### 5.2.3 Computing Adversarial Perturbations

Given the “forward” process of generating a set of frames for a given set of modification values \( \Phi \), we finally describe our approach to finding the value of \( \Phi \) that misleads the control network. We let \( D(\{I_{hf}[n]\}_f) \) denote the controller network, which takes as input a sequence of frames \( \{I_{hf}[n]\} \) in a single trajectory and generates a corresponding sequence of real-valued control signals, such as a steering angle at each time instant. Our goal is to find the value of \( \Phi \) that maximizes deviation of these outputs from those for an unmodified environment. We cast this as minimization of a loss over our \( T \) trajectories, i.e.,

\[
\Phi = \text{arg} \min_{\Phi} - \sum_{t=1}^{T} \delta \left( D(\{I_{hf}^t[n, \Phi]\}_f), D(\{	ilde{I}_{hf}^t[n]\}_f) \right), \tag{5.3}
\]

in which \( \delta(\cdot, \cdot) \) measures divergence between two sequences of control outputs. In addition, we propose a physically meaningful variation of this, where we split the \( T - 1 \) trajectories other than the one using actual control outputs into two subgroups: the one with positive and the other with negative perturbations, with both groups including the original trajectory. Note that this is meaningful in the \textit{Groups} approach for generating trajectories above, but not for \textit{Random}. After solving the two resulting problems independently, we can then choose the solution that has the highest divergence. In our experiments where the control network
outputs a sequence of steering angles, we use the absolute value of the sum of the differences between the angles as our divergence metric when we pool all trajectories together, and use the difference (for the positive subgroup) or negative difference (for the negative subgroup). We do this because we would expect that positive perturbations will be more representative when we are trying to force steering further to the right, and negative perturbations are most physically meaningful when we aim to cause sharp steering to the left.

We carry out either version of optimization iteratively using gradient descent with Adam [66] because, as shown next, we are able to compute gradients of the loss in (5.3) with respect to the modification parameters \( \Phi \). Since the controller network \( D(\cdot) \) is assumed to be a neural network (or a differentiable function), we are able to use standard back-propagation to compute gradients \( \nabla (I^f_j(n; \Phi)) \) of the loss with respect to each intensity in the images of the modified environment. The gradients with respect to the color parameters \( \{x^C_k\} \) can then be computed based on (5.2) as:

\[
\nabla(x^C_k) = \sum_{t,f} (C^t_j)^T \left( \sum_n V_k(G^t_j(n); \Phi) \nabla (I^f_j(n; \Phi)) \right).
\]

(5.4)

Computing gradients with respect to the shape parameters \( \{x^S_k\} \) requires an approximation, since the mask functions \( M(\cdot) \) are not generally differentiable with respect to these parameters. We adopt a simple local linearization approach: for every scalar parameter \( \theta \) in the shape parameters \( x^S_k \) for each figure, we discretize its range into a fixed set of equally separated values. Then, given the current (continuous) value of \( \theta \), we let \( \theta^- \) and \( \theta^+ \) represent the consecutive pair of values in this discrete set, such that \( \theta^- \leq \theta \leq \theta^+ \), and denote \( \Phi_{\theta^-} \) and \( \Phi_{\theta^+} \) the full set of current parameter values, with \( \theta \) replaced by \( \theta^+ \) and \( \theta^- \) respectively. We make the approximation that if \( \alpha \in \mathbb{R} \) such that \( \theta = \alpha \theta^+ + (1 - \alpha) \theta^- \), then \( I^f_j(n; \Phi) \approx \alpha I^f_j(n; \Phi_{\theta^+}) + (1 - \alpha) I^f_j(n; \Phi_{\theta^-}) \). Therefore, although we only use frames \( I^f_j(n; \Phi) \) with the actual value of \( \Phi \) as input to the control network, we also generate an extra pair of images \( I^f_j(n; \Phi_{\theta^-}), I^f_j(n; \Phi_{\theta^+}) \) for each frame for every element \( \theta \) of the shape parameters. We then use these frames to compute parameter gradients as:
\[ \nabla(\alpha) = \sum_{t,f,n} \nabla(I^f_t(n; \Phi)) \left( I^f_t(n; \Phi_{g+}) - I^f_t(n; \Phi_{g-}) \right), \]

\[ \nabla(\theta) = \nabla(\alpha) \times (\theta^+ - \theta^-)^{-1}. \] (5.5)

### 5.3 Experiments

Our experiments evaluate attacks against the neural network-based controller network that is included with Carla and uses only camera inputs and outputs steering angles; this network was trained using imitation learning. We run evaluations only on scenarios where this controller drives successfully without infractions in the unmodified environment.

Our experiments use 40 scenarios of driving through a stretch of road in a virtual town. Each scenario begins an episode with the vehicle spawned at a given starting waypoint, and the controller is then tasked with reaching a defined destination waypoint. The episode runs until the vehicle reaches this destination or a time-limit expires (e.g., if the car crashes). Our scenarios are of three types: (a) the expected behavior is for the car to go straight (16 scenarios), (b) veer left (12 scenarios), or (c) right (12 scenarios). In each scenario, the attacker can draw a pattern on the road with the intent of causing the car to deviate from its intended path. We consider patterns that are unions of rectangles (i.e., each “figure” in Sec. 5.2.1 is a rectangle), where the shape of each rectangle is determined by four parameters (i.e., \( x^C_k \in \mathbb{R}^4 \)): rotation, width, length, and horizontal offset.1

Each rectangle in our shape is parameterized by four values \( x^S_k = [w, l, o, \theta] \), corresponding to width, length, horizontal offset, and rotation or orientation. These parameters are defined with respect to the top-down view of a 400 \times 400 pixel “canvas” that is composited onto the road surface. Each rectangle is first drawn aligned with the \( x- \) and \( y- \) axes of this canvas to be of width \( w \) and length \( l \) pixels, and vertically centered and horizontally offset so that its left edge is at \( x = o \), and then rotated about the center of the canvas by angle \( \theta \). Portions of rectangles that lay outside the canvas after this process were clipped from the pattern. Our parameterization expands on the one originally used by (author?) [7] in two respects: by

---

1We center the rectangles vertically prior to rotation, as in [7].
searching over length $l$ instead of fixing it to the length of the canvas, and having a separate orientation $\theta$ for each rectangle rather than a common one for all rectangles.

We estimate homographies between the canvas and each frame from 60 corresponding pairs as described in Sec. 5.3, using a direct linear transform. While doing so, we ensure the matrix has the correct sign so that homogeneous coordinates of points projected in the frame have a positive third coordinate when they are visible, and a negative one when they are “behind the camera”. When compositing patterns on the frame, this allows us to retain only the portion of the pattern that would be visible from that viewpoint. The color transforms are estimated simply from the color correspondences using a least-squares fit.

We report results from optimizing shape and color parameters for different numbers of rectangles $K$, ranging from $K = 1$ (7 parameters) to $K = 5$ (35 parameters), and additionally for the single black rectangle case, also when optimizing only its shape (4 parameters). We learn these parameters with respect to the top-view coordinate frame of a canvas, that during evaluation will be passed to the simulator to be superimposed on the road (and then captured by the camera as part of the scene in all frames).

We train both BO and GradOpt in a simulated setting without any pedestrians or other cars. We then evaluate the success of the attack on actual simulations with Carla, in terms of two metrics. The first measures deviation between the paths with and without the attack pattern without pedestrians or other cars. We define deviation as

$$\text{Deviation} = \frac{1}{2T} \sum_{t=1}^{T} \min_{t'} |\tilde{W}_t - W_{t'}| + \min_{t'} |W_t - \tilde{W}_{t'}|,$$  \hspace{1cm} (5.6)

where $\tilde{W}_t$ and $W_t$ are sequences of car locations when driving with and without the attack pattern, at a fixed set of time instances defined as those when the region of the road where the attack pattern would appear is visible. Our second metric is the total infraction penalty when driving with pedestrians and other cars, as defined by the Carla Autonomous Driving Challenge [121] (for example, a lane violation carries a penalty of 2 points, hitting a static object or another vehicle of 6, hitting a pedestrian of 9, etc.). For each attack and scenario, we run 10 simulations, randomly spawning pedestrians and cars each time, and average the infraction penalty scores. Finally, while both BO and GradOpt are trained in a clear noon weather setting, we measure infractions on that setting as well as three others: cloudy noon, rainy noon, and clear sunset.
5.3.1 Attack Optimization

Proposed Method: Our approach requires two steps for every scenario: (1) collecting a set of frame sequences and calibrating them, and (2) performing gradient-based optimization. For (1), we collect frames from \( T = 3 \) trajectories; we compare different ways to generate these and the impact of varying \( T \) below. We estimate the homographies \( G^t_f \) for every frame \( f \) in trajectory \( t \) by running additional simulations with calibration patterns painted on the canvas: we run 12 simulations for each trajectory, with 5 calibration points of different colors in each simulation, and use the 60 automatically detected correspondences to estimate the homography matrix for every frame. We also learn a common set of color transform parameters for all frames in all trajectories, which we obtain by running the simulator 22 times, each time with the entire canvas painted a different high-contrast color, on the unperturbed trajectory. In addition, we collect clean frames for each trajectory. Altogether, our method calls the simulator a total of 61 times.

Once we have a set of calibrated frames, we employ gradient-based optimization (see Sec. 5.2.3). We run the optimization with four different random starts, each for 250 iterations, for a total of 1000 iterations. We begin with a learning rate of 0.1 for each start, and drop it by a factor of 0.1 after the first 100, and the first 200 iterations. Each time we drop the learning rate, we also reset the parameters to those that yielded the lowest value of the loss thus far.

Bayesian Optimization: We employ BO with the same objective as ours based on the same divergence metric, and closely follow the setting in [7]—i.e., we run the optimization for a total of 1000 iterations, of which the first 400 are random exploration. Note that while this is the same number of iterations as we use for our method, every iteration of BO requires running a full episode with the Carla simulator.

5.3.2 Results

Run-time: Recall that the training budget for both BO and GradOpt, is 1000 iterations. BO requires a simulator call at each iteration, with training times ranging 7-25 hours, depending on the scenario. In contrast, our method only calls the simulator 61 times for calibration. Ignoring the potential of parallelizing the iterations for the four random starts, GradOpt has a total running time of up to 2.5 hours per scenario, including both calibration and training.
Thus, our method affords a significant advantage in computational cost and, as we show next, is also more successful at finding optimal attack patterns.

<table>
<thead>
<tr>
<th>K</th>
<th>Group T=1</th>
<th>Group T=3</th>
<th>Group T=5</th>
<th>Group T=7</th>
<th>Random T=3</th>
<th>Random T=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-b</td>
<td>0.89</td>
<td>0.92</td>
<td>1.03</td>
<td>0.98</td>
<td>0.92</td>
<td>0.84</td>
</tr>
<tr>
<td>1</td>
<td>0.86</td>
<td>0.89</td>
<td>0.89</td>
<td>0.95</td>
<td>0.94</td>
<td>0.86</td>
</tr>
<tr>
<td>2</td>
<td>1.05</td>
<td>1.13</td>
<td>1.07</td>
<td>1.20</td>
<td>1.13</td>
<td>1.16</td>
</tr>
<tr>
<td>3</td>
<td>1.11</td>
<td>1.14</td>
<td>1.22</td>
<td>1.33</td>
<td>1.26</td>
<td>1.09</td>
</tr>
<tr>
<td>4</td>
<td>1.21</td>
<td>1.30</td>
<td>1.28</td>
<td>1.28</td>
<td>1.32</td>
<td>1.14</td>
</tr>
<tr>
<td>5</td>
<td>1.23</td>
<td>1.33</td>
<td>1.33</td>
<td>1.33</td>
<td>1.33</td>
<td>1.22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>Group T=1</th>
<th>Group T=3</th>
<th>Group T=5</th>
<th>Group T=7</th>
<th>Random T=3</th>
<th>Random T=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-b</td>
<td>3.80</td>
<td>4.25</td>
<td>4.19</td>
<td>4.43</td>
<td>3.82</td>
<td>3.98</td>
</tr>
<tr>
<td>1</td>
<td>4.18</td>
<td>5.20</td>
<td>5.54</td>
<td>4.97</td>
<td>4.73</td>
<td>4.68</td>
</tr>
<tr>
<td>2</td>
<td>3.86</td>
<td>5.16</td>
<td>5.33</td>
<td>5.85</td>
<td>4.96</td>
<td>4.88</td>
</tr>
<tr>
<td>3</td>
<td>3.79</td>
<td>6.04</td>
<td>5.62</td>
<td>6.50</td>
<td>5.28</td>
<td>4.97</td>
</tr>
<tr>
<td>4</td>
<td>5.04</td>
<td>6.35</td>
<td>5.45</td>
<td>6.00</td>
<td>5.39</td>
<td>5.39</td>
</tr>
<tr>
<td>5</td>
<td>4.17</td>
<td>6.65</td>
<td>6.29</td>
<td>6.26</td>
<td>6.54</td>
<td>5.83</td>
</tr>
</tbody>
</table>

Table 5.1: Ablation analysis of variations of GradOpt.

**Ablation Analysis of GradOpt:** First we identify the best variation of GradOpt, in terms of the choice of $T$, the choice between Random (randomly perturbing each trajectory) and Group (generating pairs of perturbed trajectories, one with positive and another with negative perturbations), and for the latter, whether we pool all trajectories in one optimization problem (Group-All) or separately optimize only positively/negatively perturbed trajectories, respectively (Group). The results in Table 5.1 show that Group has the best performance, particularly in terms of infraction penalties. Moreover, $T = 3$ yields significant improvement over $T = 1$, but further increasing $T$ does not. This shows that our approach that makes use of perturbed trajectories to counter the car’s self-correcting behavior is indeed important, and remarkably effective, requiring only 2 perturbed trajectories. Moreover, we can see that separately solving the problem with only positive, and only negative, perturbation (both including the baseline), rather than pooling these into a single objective, is important in yielding more infractions, even though there is no difference in terms of divergence. The intuition for this is that pooling only, say, positively perturbed trajectories makes them
consistent with the goal of the optimization (which is to steer sharply to the right, in that case), and the attack is better able to counter self-correcting behavior of the vehicle. Thus, we use the Group variant of GradOpt in the sequel.

<table>
<thead>
<tr>
<th>K (#Rect.)</th>
<th>Deviation</th>
<th>Infraction Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BOGradOpt</td>
<td>% ≥ BOGradOpt</td>
</tr>
<tr>
<td>1-b</td>
<td>0.85</td>
<td>0.92</td>
</tr>
<tr>
<td>1</td>
<td>0.85</td>
<td>0.89</td>
</tr>
<tr>
<td>2</td>
<td>0.79</td>
<td>1.13</td>
</tr>
<tr>
<td>3</td>
<td>0.93</td>
<td>1.14</td>
</tr>
<tr>
<td>4</td>
<td>0.70</td>
<td>1.30</td>
</tr>
<tr>
<td>5</td>
<td>0.84</td>
<td>1.33</td>
</tr>
</tbody>
</table>

Table 5.2: Average deviation and infraction penalties over all scenarios for GradOpt and BO, when optimizing parameters of different numbers of rectangles (1-b optimizes only the shape of one black rectangle) in “clear noon” weather. The % ≥ column reports the percentage of instances where GradOpt has ≥ score than BO.

Figure 5.2: Trajectory deviations induced by GradOpt and BO for 6 example scenarios.

**Efficacy of GradOpt:** Table 5.2 compares GradOpt and BO in terms of the metrics deviation and infraction penalty discussed above, computed with simulations in the same standard weather setting as used for attack optimization. We report the averages, as well as the percentage of scenarios when GradOpt outperforms BO as comparison statistics. We find that GradOpt is significantly more successful than BO, despite also being computationally less expensive as discussed earlier. It has higher average values of both deviation and infraction penalties, with the gap growing significantly for higher values of $K$—indicating that GradOpt is much better able to carry out optimization in a higher-dimensional parameter space and leverage the ability to use more complex patterns. Moreover, we find that it yields
attack patterns that are as or more successful than BO in more than 70% of cases in all settings, with the advantage rising to 82% for $K = 5$. Figure 5.2 shows some example scenarios comparing trajectory deviations induced by the attack patterns discovered by the two algorithms. Additional illustrations and visualizations are provided in the Supplement.

We take a closer look at the vulnerability of different types of scenarios by separately reporting infraction penalties for each in Table 5.3.2. In addition, we also report the corresponding infraction penalties computed in simulations without cars or pedestrians in Table 5.4. We see that scenarios where the expected behavior is driving straight are the hardest to attack, likely because they are the simplest to drive in. BO tends to achieve only a moderate infraction score in these settings, even at higher values of $K$. In contrast, GradOpt reveals that even these scenarios are in fact vulnerable when one is allowed to consider more complex adversarial patterns—achieving an average infraction penalty that is significantly higher than BO at $K = 5$. Conversely, driving right scenarios are the most vulnerable with both methods being successful even with simple patterns, with GradOpt again yielding higher deviation and more infractions.

<table>
<thead>
<tr>
<th>K</th>
<th>#Rect.</th>
<th>Straight</th>
<th>BO</th>
<th>GradOpt</th>
<th>% ≥</th>
<th>Left</th>
<th>BO</th>
<th>GradOpt</th>
<th>% ≥</th>
<th>Right</th>
<th>BO</th>
<th>GradOpt</th>
<th>% ≥</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-b</td>
<td>2.24</td>
<td>2.33</td>
<td>84%</td>
<td>4.70</td>
<td>4.15</td>
<td>76%</td>
<td>5.14</td>
<td>6.90</td>
<td>78%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2.02</td>
<td>2.81</td>
<td>77%</td>
<td>5.53</td>
<td>4.79</td>
<td>60%</td>
<td>6.11</td>
<td>8.78</td>
<td>68%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.91</td>
<td>2.73</td>
<td>91%</td>
<td>5.28</td>
<td>5.28</td>
<td>63%</td>
<td>7.33</td>
<td>8.28</td>
<td>68%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2.74</td>
<td>4.30</td>
<td>84%</td>
<td>5.47</td>
<td>5.45</td>
<td>67%</td>
<td>5.98</td>
<td>8.94</td>
<td>70%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.97</td>
<td>5.16</td>
<td>95%</td>
<td>5.00</td>
<td>4.55</td>
<td>72%</td>
<td>6.33</td>
<td>9.73</td>
<td>73%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1.69</td>
<td>4.86</td>
<td>86%</td>
<td>5.60</td>
<td>6.65</td>
<td>76%</td>
<td>7.92</td>
<td>9.04</td>
<td>76%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.3: Infraction penalties by scenario type (driving straight, left, or right) in “clear noon” conditions.

**Transferability**: We evaluate the robustness of our adversarial test generation approach by evaluating the success of generated adversarial perturbations in different climate and visibility conditions than those used for attack optimization. Table 5.5 presents results for simulations with four such climate settings, and as expected, we find that both BO and GradOpt do see a drop in penalty scores compared to the standard setting in Table 5.2. Table 5.6 and Table 5.7 reports the vehicle and pedestrian free deviation and infraction penalty scores for simulations in the different non-standard weather conditions. Nevertheless, most of the attacks induce
Table 5.4: Infraction penalties without cars or pedestrians, i.e., infraction penalties computed with only static objects, in standard “clear noon” simulations for each type of scenario.

<table>
<thead>
<tr>
<th>K</th>
<th>Straight</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Rect. BO GradOpt % ≥</td>
<td>BO GradOpt % ≥</td>
<td>BO GradOpt % ≥</td>
</tr>
<tr>
<td>1-b</td>
<td>2.25</td>
<td>1.50</td>
<td>62%</td>
</tr>
<tr>
<td>1</td>
<td>1.38</td>
<td>1.88</td>
<td>69%</td>
</tr>
<tr>
<td>2</td>
<td>0.62</td>
<td>1.62</td>
<td>94%</td>
</tr>
<tr>
<td>3</td>
<td>2.38</td>
<td>2.62</td>
<td>75%</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>2.25</td>
<td>100%</td>
</tr>
<tr>
<td>5</td>
<td>1.75</td>
<td>3.00</td>
<td>81%</td>
</tr>
</tbody>
</table>

Table 5.5: Infraction penalties over all scenarios with weather conditions different from that used for optimizing attacks (“clear noon”).

<table>
<thead>
<tr>
<th>K</th>
<th>Cloudy Noon</th>
<th>Rainy Noon</th>
<th>Clear Sunset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Rect. BO GradOpt % ≥</td>
<td>BO GradOpt % ≥</td>
<td>BO GradOpt % ≥</td>
</tr>
<tr>
<td>1-b</td>
<td>2.29</td>
<td>3.56</td>
<td>78%</td>
</tr>
<tr>
<td>1</td>
<td>3.19</td>
<td>3.61</td>
<td>82%</td>
</tr>
<tr>
<td>2</td>
<td>2.87</td>
<td>4.45</td>
<td>86%</td>
</tr>
<tr>
<td>3</td>
<td>3.06</td>
<td>5.21</td>
<td>85%</td>
</tr>
<tr>
<td>4</td>
<td>2.39</td>
<td>4.96</td>
<td>89%</td>
</tr>
<tr>
<td>5</td>
<td>3.56</td>
<td>5.87</td>
<td>83%</td>
</tr>
</tbody>
</table>

Table 5.6: Deviations in simulations over all scenarios with weather conditions different from that used for optimizing adversarial patterns (“clear noon”).

infractions, especially at higher values of $K$, with GradOpt again being significantly more successful than BO.
Table 5.7: Infraction penalties in simulations without cars or pedestrians over all scenarios with weather conditions different from that used for optimizing attacks (“clear noon”).

5.3.3 Visualization

Finally, we show the visualization of the controller’s behavior when driving with attack patterns in the same “drive straight” scenario in the main body.

For this scenario, in the simulations without vehicles and pedestrians, the pattern returned by BO is unable to cause a significant deviation in the vehicle’s trajectory as it drives across the stretch of road with the pattern painted on it. In contrast, GradOpt’s pattern is able to cause the car to veer sharply to the left in "clear noon" and "clear sunset" climate settings into the opposite sidewalk and crashes into the stationary object in Fig. 5.3.
Figure 5.3: Frames from driving simulations, without cars or pedestrians in different weather conditions, after introducing attack patterns from GradOpt (top) and BO (bottom).
Chapter 6

Location Spoofing Attacks on Autonomous Fleets

6.1 Introduction

Autonomous driving has the potential to transform mobility. A common (although by no means universal) vision for this transformation is that autonomous vehicles would come to serve in large fleets as part of ride-hailing services, ultimately obviating the need for people to own and drive cars [10, 86]. Indeed, Waymo has already begun a limited deployment of a fully autonomous ride hailing service [70]. However, given the potentially transformative impact such services may have on communities, it is imperative that we comprehensively understand their potential limitations.

One class of such limitations come in the form of security vulnerabilities, whereby a malicious party attempts to subvert the ride hailing service, either to specific ends, or simply to wreak havoc. We consider one important class of such vulnerabilities in which attackers can tamper with the vehicles’ perception of their location; we refer to these as location spoofing attacks. One means of actualizing such attacks is to use GPS spoofing which interferes with the vehicles GPS signal, causing the vehicle to have an incorrect belief about its current position [4, 102, 129, 139, 150, 167]. Attacks of this kind involve a spoofing device placed near, or in, a target vehicle. Most prior works focus on the technical aspects of implementing the attack itself. We take such GPS spoofing attacks as a given, and ask a broader systemic question: is it possible to leverage GPS spoofing attacks to significantly impact a ride-hailing service at scale with a limited number of spoofing devices?

To answer this question, we consider three threat models that involve placing a limited number of location (GPS) spoofing devices in a traffic network. In the first, the spoofing devices are
placed in a set of fixed locations (intersections) in the traffic network, and each device spoofs a fixed target location; we refer to this as the *static-static* model (static device locations with static spoofed locations). In our second model, the spoofing device locations are still fixed, but the target locations being spoofed can now be time-varying (the *static-dynamic* model). Finally, our third model allows both the devices and the locations to vary with time, for example, with devices being transported in malicious drones. We consider two possible attack goals: 1) maximizing overall delay of servicing requests, and 2) maximizing the number of failed (not completed) requests, where a failed request is one in which a passenger is dropped off at an incorrect location.

We begin with a negative result from an attacker’s perspective (positive from the vantage point of a fleet manager): even in the static-static setting, computing an optimal location spoofing attack is NP-hard, even when one targets a single vehicle and has an unlimited spoofing budget. Next, we present algorithmic approaches for each of the three threat models. For the static-static model, we propose an integer linear programming (ILP) formulation for both service delay and service failure attacks. In addition, we present a simple greedy heuristic approach for the service delay attack in this setting. We then adapt the ILP formulation from the static-static to the static-dynamic setting, blending it with a greedy heuristic approach for selecting spoofed locations. Finally, we present a deep Q-network (DQN)-based reinforcement learning approach for learning a spoofing attack strategy in the dynamic-dynamic setting.

We evaluate the proposed attack methods through extensive experiments. Our results demonstrate that attacks can indeed be highly effective, despite the negative worst-case hardness results. We show that our proposed approaches outperform several baselines, often by a significant margin, particularly as the problem size increases.

In summary, we make the following contributions:

- We systematically study the effects of GPS spoofing attacks on multi-vehicle self-driving fleets, under two types of attacking objectives (maximizing delay and maximizing number of incomplete trips) and three paradigms of spoofing capabilities (static-static, static-dynamic, dynamic-dynamic).

- We show that the attacker’s problem is intractable for each combination of spoofing capabilities and objective.
- Despite this intractability we provide efficient solutions for developing attacks in practice: an ILP formulation for the static-static case, an effective greedy heuristic for the static-dynamic case, and a deep reinforcement learning approach in the dynamic-dynamic case.

- We experimentally demonstrate the effectiveness of each attack method on a simulated traffic network of New York City (sourced from OpenStreetMap and Uber Movement), and demonstrate that even a few spoofing devices can significantly impact the performance and reliability of a large autonomous vehicle fleet.

### 6.2 Additional Related Work

In recent years extensive research, both academically and commercially, has focused on developing fully autonomous vehicles [13, 27, 1]. Many of these autonomous driving paradigms can be thought of as consisting of two systems: a low-level system which is responsible for controlling the vehicle and effectively navigating the roadway, and a high-level system which is responsible for routing the vehicle to its destination. Our focus will be primarily on the high-level system. Within this high-level navigation system GPS is the de facto means of
determining vehicle location. When navigating via GPS location information, the vehicle possesses a receiver which accepts GPS signals which indicate the vehicle’s current location. Recent work has demonstrated that the GPS signal received by the vehicle can be maliciously altered by spoof devices [167, 4, 28, 4, 102, 129, 139, 150, 167]. In GPS spoofing, the attacker transmits fabricated GPS signals with greater signal power than the authentic ones, thus causing the victim receiver to lock onto the attacker’s signals, rather than the authentic signal, and resolve positions controlled by the attacker. GPS spoofing has been proven feasible theoretically [31] and empirically [167, 129]. More importantly, it has been demonstrated that autonomous driving vehicles are vulnerable to GPS spoofing attacks, such as Tesla cars [62], in the physical world. Further, [167, 61] has shown that modern spoofers are cheap to manufacture.

Another orthogonal line of research has investigated the capability of using GPS spoofers to reroute autonomous vehicles. In particular, [167] has developed a spoofing attack which is capable of successfully rerouting a victim car such that it arrives at an incorrect destination. This work conducted physical experiments in New York City in which they successfully rerouted a vehicle and reached a location that is different from its destination, thus demonstrating that GPS spoofing attacks are physically realizable. In their design, the attackers knows the vehicle’s destination, the vehicles location at each time timestep, and the vehicle’s routing algorithm. Spoofers are placed at every intersection and the attacker designs a signal for each spoofer such that when the vehicle’s routing system uses the designed signal, the vehicle is rerouted to the attacker’s target location. These and similar GPS spoofing attacks typically impose two constraints: first, they require a spoofing device to be close to the target vehicle, and second, they require the spoofed location to be reasonably close to the vehicle’s true location. The latter constraint is intended to create subtle attacks, which are more difficult to detect.

While GPS spoofing attacks have been demonstrated to consequentially impact autonomous driving, prior work has only considered single-vehicle attacks. In contrast, we consider attacks on a multi-vehicle ride-hailing fleet involving multiple spoofing devices. Our attacks share some structure with variants of multi-agent routing problems [67, 132, 136], although many technical details differ from our problem structure differ.
6.3 Model

We begin with a model of a ride-hailing system, in which $K$ vehicles traverse a traffic network $G = (V, E)$, where $V$ is the set of nodes representing intersections and $E$ is the set of edges connecting the intersections. We use the set $\mathcal{N}(i)$ to denote the neighbors of node $i \in V$. Each edge $e_{ij} \in E$ in the traffic network $G$ is associated with a travel time $c_{i,j}$, which we normalize without loss of generality to be in $[0, 1]$. We identify each vehicle with an index $k \in \{1, \ldots, K\}$. Vehicle $k$ is associated with a current location $s_k$ and a destination $d_k$. Further, we assume that each vehicle $k$ follows the shortest path from any location $v \in V$ to its destination $d_k$. We assume that both $G$ and the edge costs are fixed (e.g. the traffic of each street is roughly invariant over time), and we can therefore pre-compute a shortest path from each starting point and destination.

Our model of spoofing effect on the vehicle routes is based on the model proposed by Zeng et al. [167], who constructed a physically realizable spoofing attack which effectively manipulated a single vehicle. In this model, the attacker chooses the intended effect of a spoofing device on a target vehicle with the objective of inducing a desired behavior, e.g. maximally deviating from the target destination. In our setting, we focus on a fleet of autonomous vesicles, each of which traverse the network that may be affected by the same spoofing device. Consequently, the same spoofing device and effect will be reflected in any vehicle that comes within the proximity of the device. We capture the behavior of each such vehicle by a matrix $M$, where $M_{i,j} = \alpha$ if a vehicle at node $i \in V$, heading to destination $j$, will take action $\alpha$, which can be a high-level action such as which next road segment to take, or a low-level action, such as a steering angle. In our experiments, we focus on the latter. Note that the matrix $M$ is independent of the identity of the vehicles (indeed, it can also represent any deterministic behavior of vehicles, not merely shortest paths.\footnote{Our results and attack framework can be applied to any routing deterministic routing system which is 1) agnostic to vehicle id and 2) assumes that congestion is static.}

As in Zeng et al., we assume that the attacker knows the network $G$, edge costs $c_{i,j}$ and vehicle behavior matrix $M$. These are mild assumptions: $G$ is typically public knowledge, $c_{i,j}$ can be obtained in nearly real-time from mapping applications, and $M$ is vehicle agnostic and can be pre-computed by the attacker. In addition, the attacker knows the current state of the $K$ fleet vehicles, for example, as a consequence of a separate cyber-attack on the fleet management service.
The attacker is endowed with a collection of $B$ spoofing devices. Each device can be placed at a location $i \in V$, and spoof an alternative target location $j \in V$ chosen by the adversary. As a consequence of such an attack, a vehicle at location $i$ erroneously perceives that it is actually at location $j$. It will be useful to represent the attacker’s decision by a binary variable $x$, where $x_{i,j} = 1$ iff there is a location spoofing device placed at location $i$ which targets location $j$.

In general, we allow the attack to vary with time, in which case $x_t$ will denote the attacker choice at time $t$. In particular, we consider three attack settings that represent three distinct attack capabilities, in increasing order of strength:

1. **static-static**: both the spoofing device and target locations are fixed (independent of time),

2. **static-dynamic**: spoofing device locations are fixed, but target locations can vary with time,

3. **dynamic-dynamic**: both the spoofing device locations and target locations can vary with time.

The former two threat models simply involve placing the spoofing devices at target intersections. The third requires an additional capability that these devices are placed on adversarial mobile vehicles, such as cars or drones. In order to capture any difference in the time scale of vehicle and spoofing device motion, we assume that the devices can change location every $t$ steps for $t \geq 1$ (i.e., they are no faster, but could be somewhat slower, than the vehicles).

We consider two attack objectives:

1. **service delay attack**: maximize the total travel time of the vehicles in the fleet, subject to the constraint that requests are ultimately correctly completed (that is, the rider is not dropped off at the wrong destination), and

2. **service failure attack**: maximize the number of requests in which the rider is dropped off at the wrong destination.

In the case of the first attack, the constraint that each request is ultimately completed serves as a form of stealth, as we also suppose in this context that spoofing devices are on for a
finite time duration, and turned off thereafter. The second attack, in contrast, entails vehicles actually believing that they had reached their target destinations, but in fact they have reached locations in which a spoofing device is currently active and spoofing to their dropoff location.

6.3.1 Hardness

We begin by first noting that the spoofing problem is computationally intractable in general.

Theorem 6.3.1. Maximum delay is NP-hard even for a single fleet car, a single request, unlimited spoofers and uniform congestion for both the online and offline settings, as well as for each of the three spoofing paradigms.

Proof. We can reduce from Hamiltonian cycle. Let \( G \) be an unweighted connected graph. An instance of our problem can be constructed by assigning every edge in the graph weight 1 and creating two nodes called \( s \) and \( d \) where \( s \) has an outgoing edge with weight 1 to each node in \( G \) and each node in \( G \) has an outgoing edge with weight 1 to \( d \). Let \( s \) and \( d \) be the fleet car’s starting location and destination respectively. A second set of nodes and edges is created such that for each edge \((u, v)\) in \( G \) a new node \( u' \) is created and the edge \((u', d)\) is created to correspond to \((u, v)\) so that if the vehicle is at node \( u \), but has perceived location \( u' \) it will move to \( v \) since it thinks it is moving to \( d \). Since congestion is uniform, the spoofing budget is unlimited, and each edge \((u, v)\) has a corresponding edge \((u', d)\) the attacker has arbitrary control over the vehicle’s movement and thus the attacker is selecting the vehicle’s path with the constraint that the path must start at \( s \) and end at \( d \).

Suppose that a solution to Hamiltonian cycle on \( G \) is given, then starting from some node \( u \) and ending at some node \( v \) we have a path which reaches every node in \( G \) exactly once. If the adversary where to choose to send the vehicle from \( s \) to \( u \) and then along the Hamiltonian cycle solution path to \( v \) and then from \( v \) to \( d \) the vehicle’s travel time would be \( n + 1 \). Since each node can be visited at most once by the car, otherwise the car will never reach its destination, and traveling between any two neighbor nodes has cost 1, the maximum travel time the car could have is \( n + 1 \). Therefore given a solution to Hamiltonian cycle, the spoof attack with the largest travel time can be created. If instead we had an optimal spoofing attack, then since traveling between any two nodes has cost 1 and the agent cannot repeat
nodes, otherwise they will not reach \( d \), the number of nodes visited is equal to \( t + 1 \) where \( t \) is travel time. Since the agent can only reach nodes \( s, d \) and all nodes in \( G \), and must reach nodes \( s \) and \( s \), if travel time is \( t \) then the agent reached \( t - 1 \) nodes in \( G \). Thus if the spoof attack returns travel time \( n + 1 \) then the vehicle reaches \( n \) nodes in \( G \) exactly once, and \( G \) has a Hamiltonian cycle. Therefore, given a solution to GPS spoofing a solution to Hamiltonian cycle can be found, and given a solution to Hamiltonian cycle a solution to GPS spoofing can be found.

Despite the hardness result, we next proceed to develop effective algorithms for both static and dynamic spoofing.

6.3.2 Approach: Design of GPS Spoofing Attacks on Autonomous Fleets

Static Location of Spoofing Devices and Spoofing Targets

We begin by considering static-static attacks in which \( B \) spoofing devices are placed at fixed locations in the traffic network, and each is configured to spoof a fixed target location. We first develop an integer linear programming approach when the attack goal is to maximize service delay (with the constraint that the destination is ultimately reached). Subsequently, we show that the second problem of maximizing service failure is tractable, and present an efficient algorithm for it.

**Service Delay Attack** Recall that when a spoofer and a fleet vehicle are on the same node in the graph, the fleet vehicle will perceive its location to be a different node in the graph specified by the spoofer. Our goal is to find the placement and effect of \( B \) spoofers, resulting in the maximum delay to the \( K \) fleet vehicles. We adopt two constraints on GPS spoofing attacks proposed by Zeng et al. [167] which ensure physical realizability. The first constraint is on the maximum spoofing distance from the device location. The second constraint prevents the action \( \alpha \) induced by the spoofing device from being infeasible in the *actual* location that the vehicle is in, such as turning left when no left turn is available.
Recall that $x$ is a matrix representing spoofing device locations (rows) and targets (columns); since both are static, $x$ does not depend on time. Let $\tau(i, j, x)$ denote the travel time from location $i$ to destination $j$ if spoofing devices are placed and configured according to $x$. Then the attacker’s goal is to solve the following optimization problem: $\max_x \sum_{k=1}^{K} \tau(s_k, d_k, x)$, that is, to choose spoofing device placement and configuration $x$ that maximizes total travel time for all vehicles from their starting locations to their respective destinations. Note that travel time is also implicitly a function of the movement tensor $M$ which represents actions vehicles take in each location and for each possible destination. Critically, when we spoof locations, the net effect is that the vehicles are choosing their actions using an induced movement tensor $M'$, where the entries in $M$ corresponding to a location $i$ are replaced with entries corresponding to a spoofed location $j$ (when a spoofing device is placed at $i$ and targets $j$).

We propose an ILP formulation for this problem. In this formulation, we construct the movement matrix induced by the spoofing devices $M'$, as well as the associated shortest paths encoded by $y$ for each vehicle, as an explicit function of the binary spoofing decision matrix $x$, where $x_{i,j} = 1$ if we place a spoofing device at location $i$ and spoof location $j$, and $x_{i,j} = 0$ otherwise. As mentioned above, we constrain spoofing distance from the location of the spoofing device, and impose a constraint that a vehicle action induced by the GPS spoofing device is feasible in its actual location. We encode these constraints using a 3 dimensional tensor $F_{\{V,V,A\}}$, and construct this feasibility tensor $F$ beforehand, where $A$ is the set of possible vehicle actions. Specifically, $F_{i,j,\alpha} = 1$ when both (1) the distance between location $i$ and location $j$ is within the maximum spoofing radius of a spoofer, and (2) it is feasible to take action $\alpha$ at location $i$. However, if either constraint doesn’t hold, $F_{i,j,\alpha}$ is set to be $-1$. Let $y^k_{i,j}$ be binary variables that represent whether the edge $(i, j)$ is traversed by the vehicle $k$. Armed with this notation, formulation (6.1) gives an integer program that computes the optimal static-static attack.
The objective maximizes the total travel cost (time) incurred by all vehicles. Constraints (6.1b)-(6.1d) are standard network flow constraints. Constraint (6.1e) effectively copies the entries from $M$ to $M'$ corresponding to the spoofed locations (the first term on the right-hand-side), and leaves unspoofed entries as given by $M$ for locations without a spoofing device (the second term on the right-hand-side). Shortest path actions $y$ are then computed using the movement tensor $M'$ induced by the attack in Constraints (6.1f) and (6.1g). Finally, Constraints (6.1h) include the constraint on the maximum number of spoofing devices, and ensure that only one location can be spoofed by any device.

While the ILP has constraints which are bilinear, linearization is straightforward (given in the Supplement).

**Service Failure Attack** To maximize the number of failed requests is to maximize the number of vehicles that mistakenly drop off a passenger at the wrong destination. This clearly entails two features of any spoofing strategy: 1) a device must be placed in a location which some vehicle reaches at some point during its commute, and 2) the device spoofs a
destination for these vehicles. This suggests the following simple SF-GREEDY algorithm for choosing spoofing device locations and targets:

1. Partition the vehicles into groups such that each group \( G_{ld}, l \neq d \), contains all the vehicles having two properties: a) they share the destination \( d \) and b) their paths to destination induced by \( M \) intersect at location \( l \).

2. Choose \( B \) largest groups \( G_{ld} \) with distinct locations \( l \) (that is, if there are two groups \( G_{ld} \) and \( G_{ld}' \) for some location \( l \), we choose the larger group and omit all others). For each chosen group, place the spoofing device in location \( l \) with a spoofing target \( d \).

This algorithm clearly runs in polynomial time. The next result, which is proved in the Supplement, shows that this algorithm yields an optimal solution.

**Theorem 6.3.2.** SF-GREEDY yields an optimal solution to the service failure attack in the static-static setting.

**Proof Sketch.** First, note that if we have multiple vehicles traversing \( l \), any spoofing sequence starting with the device placed in \( l \) will group them together, since we constrain that only a single location can be spoofed by any spoofing device (and no more than one such device can be placed at an intersection). Thus, we would ultimately only cause service failure for the the largest subgroup of vehicles who traverse \( l \) that share a destination. But this means that all of these vehicles can simply be directly pointed to their destination by the spoofing device at \( l \). Consequently, there is always an alternative optimal solution in which each spoofing device at location \( l \) simply points to the destination \( d \) of the largest group that shares the destination, yielding the structure of the solution in SF-GREEDY.

**Static Location of Spoofing Devices with Dynamic Spoofing Targets**

In the case of static spoofing device location, but dynamic spoofing (effect can update over time), we can no longer directly apply either the ILP when the objective is to maximize delay, nor the greedy algorithm above if the objective is to maximize service failure.

We propose the following heuristic for this case when the objective is to maximize delay. First, we compute optimal spoofer placement via the the ILP (6.1), assuming that target locations
Dynamic Location of Spoofing Devices and Spoofing Targets

In the fully dynamic case, instead of making only a single decision regarding spoofing location, the attacker now makes a sequence of decisions within a time window. At each decision time, the attacker can change both the spoofing effect and spoofing location by moving a spoofing device in one of a collection of locations \( L \) (e.g., up, down, left, right, and no change, if the traffic network is a grid). This can be implemented, for example, if the spoofing devices are located in malicious vehicles or drones. We will refer to a spoofing device (which is now mobile) as an agent in this section. The underlying problem in this setting is a dynamic (discrete time) decision about the location and spoofing target of each agent (each spoofing device) through a given horizon, and extremely complex combinatorial optimization problem. We approach this problem using a multi-agent reinforcement learning (MARL) paradigm. Specifically, each agent independently learns to decide which location to visit and what the spoofing effect (i.e., spoofing target location) is at each decision time. A key challenge in applying MARL in our setting is how to design individual agent rewards, as well as the state space representation, to enable effective learning. Next, we present our solution to both of these issues.

Rewards  We designed delay attack reward and service failure reward for each of the attacking objectives discussed in Section 6.3. In the attack aiming to maximize the overall delay of the fleet, which we denote by \( \eta \), the objective is

\[
\eta = \sum_k \sum_t (\tau_{t}^{k} - \tau_{t}^{k}),
\]

where \( \tau_{t}^{k}\) is the travel time after the attack while \( \tau_{t}^{k}\) is the travel time before the attack. The incremental delay at time \( t \) of car \( k \) is \( \rho_{t}^{k} = \eta_{t}^{k} - \eta_{t}. \) This incremental delay of car \( k \) is caused by spoofer \( j \), if the spoofer \( j \) is directly next to car \( k \) (i.e., they share the intersection).
We use the notation $z_t(k, j)$ as a delay indication variable for agent $j$, $z_t(k, j) = 1$ if agent $j$ shares the intersection with car $k$ (i.e., is able to spoof the location for this car). We use the incremental delay caused by agent $j$ as the reward for this agent at time $t$, i.e.,

$$\rho^j_t = \sum_k z_t(k, j) \cdot \rho^k_t$$

In the case of service failure attack, the reward is simpler: if the agent succeeds in causing service failure in step $t$, it receives the reward of 1; otherwise, it receives reward of 0.

**State Representation** In the dynamic-dynamic case, we construct state representation based on two types of information relating to the traffic network: *invariant*, consisting of information which does not change from one time-step to another (e.g., traffic network structure), and *real-time*, consisting of information which changes over time (e.g., vehicle locations).

This representation is deigned to be applicable to any transformer-based architecture.

The *invariant information* is represented in the form of node features, which capture important and static aspects of each node $i \in V$. We refer to the matrix of all node features as $\xi$, where $\xi_i$ denotes the features of node $i$.

Specifically, each node $i$ is mapped to two types of features. First, we include information about costs (e.g., congestion) of all out-edges incident to $i$, which in our case of grid networks includes (up to) four incident edges. This implicitly assumes (as does our model above) that spoofing has no effect on edge costs (e.g., when the size of the fleet is small relative to overall traffic).

Second features of node $i$ include movement features consisting of $M_{i,j,d}$, flattened as a vector, over all values of destinations $j$ and movement directions $d$.

*Real-time (or time-evolving)* features consist of features corresponding the vehicles’ behavior, and action of other agents (i.e., other spoofing devices), at each time $t$. Recall that in the static-static setting, $M'$ denoted the movement tensor of vehicles after spoofing. Here, as
this quantity now changes over time, we denote the corresponding tensor by \( M'_t \); this is our first set of real-time features.

Next, we associate the following real-time features with each vehicle \( k \):

1. \( \delta^k_t \), which is the vehicle’s current destination (which we now allow to evolve in time); this allows us to know which portion of the movement tensor \( M' \) the vehicle is effectively using to determine its next action,

2. \( h^k_t \), the time that the vehicle requires to complete its current action (i.e., to travel to the next node); again, note that this generalizes our model by allowing this to vary by vehicle and edge, as well as in time,

3. \( l^k_t \), the current location of the vehicle, and

4. \( M'_t \), the current move instruction, and

5. \( \zeta^k_t \), the vehicle’s path induced by the current positions of spoofing devices and their spoofing targets.

Finally, we include as a feature the location of the spoofing agent \( j \) at time \( t \), denoted by \( l^j_t \). We denote the real-time features by \( s_t \).

### 6.3.3 Experiments

For our experiments, we use a Manhattan, NY traffic network obtained using OpenStreetMap [25]. Following the convention in [167], we construct a directed graph \( G = (V, E) \) to represent the road network. Each node represents an intersection, and has a unique id as well as coordinates in the constructed graph. Adjacent nodes \( i, j \in V \), which are intersections connected by a road, are themselves connected by a directed edge \( e_{i,j} \in E \). Additionally, each pair of edges \( e_{ij}, e_{jk} \) has a turning angle \( \alpha_{ijk} \), which gives the angle required for a car to turn from intersection \( j \) (when arriving from road \( e_{ij} \)) onto road \( e_{jk} \).

In addition to the purely geographic information about the traffic network obtained from OpenStreetMap, we add congestion to each edge \( e_{ij} \) using real traffic congestion data for each road segment obtained from Uber Movement [141], an open source platform that provides
real time traffic flow data collected by Uber users. Specifically, we use the average congestion for each Friday at 5 pm in March, 2020; this is

the most recent traffic data available for New York City. Uber Movement and OpenStreetMap use the same convention to label both locations and road segments, and the data from these can therefore be directly integrated.

We conduct experiments in a geographic area centered at 350 5th Ave, Manhattan, New York, with a radius of 500 or 1000 meters; Figure 6.1 gives this network (black lines) for a 1000m radius.

Finally, we build a traffic simulator based on the downloaded traffic network and the traffic flow information. In this simulator, requests are assigned to fleet vehicles by the routing center, thereby defining each vehicle’s destination as either the pickup or dropoff location of the assigned request. Requests are spawned uniformly at random across the network.

**Evaluation Metric:** We use delay ratio to evaluate the effectiveness of the attack, defined as

\[ \omega = \frac{\tau' - \tau}{\tau} \]  

(6.2)

where \( \tau \) is the original travel time and \( \tau' \) the travel time in the presence of malicious spoofing. We conduct experiments varies from 1 to 20 fleet sizes with a spoofing budget of 1, 5, or 10.

<table>
<thead>
<tr>
<th>Spoofing Radius</th>
<th>Spoofing Budget</th>
<th>#Target Cars</th>
<th>Travel Time</th>
<th>Proposed Delay Ratio</th>
<th>Greedy Delay Ratio</th>
<th>Random Delay Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>200</td>
<td>0.11</td>
<td>0.11</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>262</td>
<td>0.02</td>
<td>0.02</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
<td>262</td>
<td>0.5</td>
<td>0.3</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>10</td>
<td>242.7</td>
<td>0.3</td>
<td>0.3</td>
<td>0.01</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>5</td>
<td>262.2</td>
<td>0.9</td>
<td>0.56</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>10</td>
<td>242.7</td>
<td>0.75</td>
<td>0.34</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>242.7</td>
<td>0.93</td>
<td>0.5</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>20</td>
<td>252.79</td>
<td>0.72</td>
<td>0.37</td>
<td>0.02</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td>242.7</td>
<td>1.1</td>
<td>0.7</td>
<td>0.01</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>20</td>
<td>252</td>
<td>0.9</td>
<td>0.75</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Table 6.1: The average travel time in unmodified driving environment and delay ratio induced by spoofers in the *static-static* case in a *dist-1000 meters* traffic network. The travel time unit is second.
**Baselines:** We compare our method to two baseline methods. First, *random spoofing*, where the locations and targets of spoofing devices are chosen randomly (in the dynamic settings, these are chosen randomly in each time step). The second baseline is *greedy spoofing*. In the static-static case, greedy spoofing consists of iteratively choosing both the location and target of each spoofing device to maximize the marginal increase in delay. In the static-dynamic case, spoofing devices are placed at locations that maximize the number of intersecting vehicle paths, while the spoofing targets (chosen at every time step) are selecting to maximize the increase in delay. In the dynamic-dynamic case, each spoofing device moves greedily at each iteration to minimize the distance to the closest non-spoofed fleet vehicle, while spoofing effects are chosen to maximize the marginal increase in total delay.

When using the random spoofing strategy in the dynamic-dynamic case, the attacker randomly moves the spoofers and randomly selects their spoofing effect.

**Results**

**Static-Static Case**  Recall that in the *static-static* case, both the locations and targets of spoofing devices are fixed. In the modified driving environment, if a GPS spoofing device is placed at location $i$ and spoofs location $j$, any fleet vehicle passing through $i$ will receive the modified GPS signal, thus perceiving its current location as $j$. This discrepancy between physical and perceived location may result in a detour from a vehicle’s intended path. Such detours may cause the car to arrive at its destination late, or be unable to complete the request.

We begin by showing the results for the maximum-delay attack, in which the attacker aims to maximize the fleet’s average delay while ensuring that the requests are still completed. Table 6.1 shows the results on different traffic network sizes. In this table we see that even with a single spoofer, delay attacks can successfully increase the fleet’s average travel time. As to be expected, delay ratio decreases as the number of fleet vehicles relative to the number of spoofers, increases. Moreover, we see that our proposed method outperforms random in all cases, and outperforms greedy when the spoofing budget is greater than 1 (note that for $B = 1$ greedy is optimal).
Table 6.2: The average travel time in unmodified driving environment and delay ratio induced by spoofers in the static-dynamic case in a dist – 500 traffic network.

**Static-Dynamic Case**  In the static-dynamic case, the attacker can update the spoofing effect over time. We propose a two-step approach to solving this case (The Algorithm is deferred to the Appendix). First we run static-static ILP formation to get location of each spoofer. Second, at each timestep we greedily select the spoof effect, for each spoofer, which results in the maximum increase to travel. Table 6.3 shows delay ratio for different numbers of victim cars, spoof budget, network size. We defer the results of network size 500 to the appendix. Our method outperforms random selection significantly in every case.

**Dynamic-Dynamic Case**  In the dynamic-dynamic case, the attacker is able to move the spoofing devices and change the spoofing targets at each timestep. A problem instance is specified by 1) the starting location of each agent (spoofing device), 2) the starting location of each fleet vehicle, and 3) the pickup and dropoff locations of the requests. We refer to the combination of these three elements as the configuration of the problem.

We train on examples with random problem configurations, and evaluate our proposed method and the baseline methods, on a randomly generated test set of 500 problem configurations. We make use of the reinforcement learning paradigm known as Target Deep Q Networks in which a deep neural network is used to approximate the Q-values of each state action pair. Here a state, the representation (described in Section 6.3.2) consists of the traffic network,
Table 6.3: The average travel time in unmodified driving environment and delay ratio induced by spoofers in the static-dynamic case in a \( \text{dist} - 1000 \) traffic network.

<table>
<thead>
<tr>
<th>Spoofing Radius</th>
<th>Spoofing Budget</th>
<th>#Target Cars</th>
<th>Travel Time</th>
<th>Proposed Delay Ratio</th>
<th>Random Delay Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>200</td>
<td>0.4</td>
<td>0.01</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>262</td>
<td>0.22</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>200</td>
<td>0.5</td>
<td>0.01</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>5</td>
<td>262</td>
<td>0.28</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
<td>262</td>
<td>0.35</td>
<td>0.04</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>10</td>
<td>242.7</td>
<td>0.92</td>
<td>0.06</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>5</td>
<td>262</td>
<td>1.9</td>
<td>0.05</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>10</td>
<td>242.7</td>
<td>1.2</td>
<td>0.09</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>242.7</td>
<td>0.17</td>
<td>0.03</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>20</td>
<td>252.79</td>
<td>0.16</td>
<td>0.02</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td>242.7</td>
<td>0.17</td>
<td>0.03</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>20</td>
<td>252</td>
<td>0.16</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Table 6.4: The delay ratio in the dynamic-dynamic case in a \( \text{dist} - 500 \) traffic network with induced by spoof devices with spoofing radius 1.

<table>
<thead>
<tr>
<th>Spoofing Budget</th>
<th>#Target Cars</th>
<th>Travel Time</th>
<th>Proposed Delay Ratio</th>
<th>Greedy Delay Ratio</th>
<th>Random Delay Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>200</td>
<td>0.90</td>
<td>0.89</td>
<td>0.03</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>262</td>
<td>2.0</td>
<td>1.2</td>
<td>0.09</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>242.7</td>
<td>1.11</td>
<td>0.67</td>
<td>0.05</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>252</td>
<td>1.0</td>
<td>0.78</td>
<td>0.09</td>
</tr>
</tbody>
</table>

Table 6.5: The delay ratio in the dynamic-dynamic case in a \( \text{dist} - 1000 \) traffic network induced by spoof devices with spoofing radius 1.
requests, fleet locations, and adversarial car locations. Each action consists of both a direction for the adversarial device to move as well as a spoofing target. Stochastic gradient decent is used to update the model. We provide full implementation details and hyperparameter choices in the Supplement.

We see that in all cases, our proposed model outperforms random and greedy. In particular, as the number of agents and fleet vehicles increases, the proposed method outperforms greedy by an increasing margin. This is to be expected as the dynamics of the problem become more subtle as the number of agents and fleet vehicles increases. Moreover, note that spoofing in the dynamic-dynamic case is more effective (causing larger delay) than the static-static case. The ability to dynamically update the spoofer device locations and targets greatly improves the efficacy of the attacks.

Finally, we evaluate our dynamic-dynamic attacks in an online setting. In this setting, new requests are dynamically spawned over time by uniformly selecting a pickup and dropoff location in the traffic network. Fleet assignment to requests is also dynamically updated once a vehicle has completed its current request. In our simulations, we adopt the assignment strategy proposed by [63]. On average, each car takes 50 requests. Complete details of the online setting are provided in the Supplement. As shown in Table 6.5, the delays induced by our RL-based attacks in the online setting are comparable to those induced when vehicle requests (i.e., destinations) are statically assigned. This corresponds to a 100-180% increase in delay.

### 6.3.4 Service Failure Attack

Next, we shows the algorithms and additional experiments results for Service Failure Attack in two cases, static-static case, static-dynamic case.

**Algorithm 1** Optimal Static-Static Service Failure Attack

```plaintext
for i in [n] do

cross_i = # of vehicles traversing i with same dropoff.

Select B location with highest value of cross_i, spoof from i to dropoff
```

66
**Algorithm 2** Static-dynamic service failure algorithm

spooferLocations ← static-static ILP solution \( \text{for } t \leq \text{maxTimeSteps} \)

\[ \text{end} \]

\[ s \in \text{spoofers} \]

\[ \text{s.eff} ← \arg \max_{v \in V} \text{delay}(\text{spoofers}|s.eff = v) \]

---

**Static-static Case**

We compare the results with random selection. In Tables 6.6 an 6.7, we see that the number of rides failed by our method is significantly larger than random selection. Moreover, we see that, on average, roughly half the requests can be failed. As the number of fleet vehicles increases, the number of failed requests decreases relative to the number of spoofers.

**Static-dynamic Case**

In the static-dynamic case, the attacker can update the spoofing effect over time. In this case, In this approach the greedy selection criterion is the effect which maximizes the increase to delay. This approach is formalized in Algorithm 2.

We compare our method with random selection. Tables and 6.9 shows that our method outperforms random selection significantly in every problem settings.
<table>
<thead>
<tr>
<th>Spoofer Budget</th>
<th>#Target Cars</th>
<th>Optimal Service Failure Ratio</th>
<th>Random Service Failure Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>0.24</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>0.18</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1.00</td>
<td>0.04</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>0.59</td>
<td>0.09</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>0.39</td>
<td>0.06</td>
</tr>
</tbody>
</table>

Table 6.6: The service failure ratio induced by spoofing devices with spoofing radius 1 in the static-static case in a dist-500 traffic network.

<table>
<thead>
<tr>
<th>Spoofer Budget</th>
<th>#Target Cars</th>
<th>Optimal Service Failure Ratio</th>
<th>Random Service Failure Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>0.21</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>0.12</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1.00</td>
<td>0.02</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>0.53</td>
<td>0.01</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>0.21</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Table 6.7: The service failure ratio induced by spoofing devices with spoofing radius 1 in the static-static case in a dist-1000 traffic network.
<table>
<thead>
<tr>
<th>Spoofers</th>
<th>#Target Cars</th>
<th>Optimal Service Failure Ratio</th>
<th>Random Service Failure Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>0.44</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>0.43</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1.00</td>
<td>0.06</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>0.96</td>
<td>0.06</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>0.79</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Table 6.8: The service failure ratio induced by spoofers in the static-dynamic case in a dist-500 traffic network.

<table>
<thead>
<tr>
<th>Spoofers</th>
<th>#Target Cars</th>
<th>Optimal Service Failure Ratio</th>
<th>Random Service Failure Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>0.39</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>0.32</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1.00</td>
<td>0.02</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>0.82</td>
<td>0.01</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>0.62</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Table 6.9: The service failure ratio induced by spoofers in the static-dynamic case in a dist-1000 traffic network.
Part III

Defense Mechanism: Developing Robust Image Recognition Model and Autonomous Driving System
Chapter 7

Patch Defense via Constrastive Adversarial Semantic Meaning

7.1 Introduction

Deep learning models have achieved remarkable success in various fields, including image classification, speech recognition, and natural language processing. However, these models are vulnerable to adversarial attacks, where malicious actors can exploit the model’s weaknesses and manipulate its output by introducing imperceptible perturbations into input data. Among these attacks, the adversarial patch attack has gained particular attention, as it focuses on manipulating images by adding small, carefully crafted patches to the original image, resulting in misclassification. As the deployment of these models in safety-critical applications increases, the need for developing effective defense mechanisms against such adversarial attacks becomes more pressing. In this paper, we present a novel detection-based approach to create models which are robust to patch attacks (AdvSemSheild). This approach works in two stage: first, the model predicts regions in a given image which correspond to adversarial patches and ablates these regions, second, the model then uses the semantic meaning present in the ablated patch to better understand the image’s true class. The latter is done by analysing the semantics of the patch and predicting which true class is most likely to be vulnerable to a patch with those semantics.

Additionally we propose a novel attack method that explicitly targets upstream detectors in addition to downstream classifiers. The unique loss function of this attack allows the attacker to reduce detection efficacy, leading to more adversarial pixels permeating the detector attacker with increased control over the model’s inference. We demonstrate the effectiveness of our attack in comparison to existing patch attack methods through experimental results.
and show that this attack can be used to craft models with increased robustness. Further, we explore the relationship between an adversarial semantic meaning and its success rate and represent the attack strategy using the adversarial patch’s semantic meaning.

Our evaluation of AdvSemSheild on three state-of-the-art classification models and three benchmark datasets demonstrates significant improvements in both clean accuracy and robust accuracy compared to state-of-the-art baselines. In addition to superior performance over these baselines, our approach offers a new perspective on how to improve model robustness to patch attacks. Unlike pixle-wise attacks, patch attacks possess the unique property that the attacker’s strategy, namely the patch itself, can be identified and analyzed in order to better understand the nature adversarial examples.

**Contributions:** In this work, we make the following key contributions:

- We introduce a paradigm to learn the semantic meaning of an adversarial patch, and use this paradigm to develop a novel defense strategy (AdvSemShield) which uses patch semantic meaning to attain superior robustness.

- We establish a relationship between the semantic meaning of a patch attack and the success of that attack, demonstrating that the attacker must include high semantic meaning in order for attacks to be successful.

- We propose a new attack that explicitly targets upstream detectors in addition to downstream classifiers, broadening the scope of adversarial patch attacks.

### 7.2 Additional Related Work

Adversarial patch attacks he adversarial patch attack [9, 65, 163] against image classification models aims to induce test time misclassification have escalated in prominence in recent years, significantly impacting the performance of deep learning models [9]. These attacks strategically manipulate models by integrating subtly-crafted patches into original images, subsequently leading to incorrect predictions. Notably, these attacks are physically realizable [90], [169], [65], meaning that the adversarial patch can be printed and physically attached to a targeted object, underscoring the tangible, real-world implications of these attacks.
To strengthen vital computer vision systems, concerted efforts have been made to develop certifiably robust defenses against adversarial patches [19, 80, 99, 154, 170]. These defense strategies aim to provide a certifiable guarantee for making accurate predictions on select images. In this vein, [158] pioneered the use of PatchGuard, a network that utilizes a small receptive field and outlier masking. Building on this, [155] proposed a defense mechanism that leverages an ensemble and an exhaustive masking technique to identify the location of the patch. It’s important to note that both methods require prior knowledge of the size of the attack patch.

Recently, to address these limitations, [159] proposed the use of a segmentation model as a detector. This innovation led to a substantial improvement in empirical performance. Such learning-based detectors are trained using adversarial training. Adversarial training, a technique that boosts the resilience of machine learning models against adversarial attacks, involves training on perturbed inputs [49], [96], and [140]. However, the efficacy of adversarial training hinges on the effectiveness of the detector employed. Our work belongs to this line of research, focusing on improving the robustness of image recognition model when faced with challenges stemming from an imperfect detector.

### 7.3 Preliminaries

**Image recognition** In our image recognition framework, we consider the image space denoted by $X \in [0, 1]^{W \times H \times C}$, where each image has a width $W$, height $H$, and $C$ channels. The pixel values are rescaled to the range $[0, 1]$. The label space is denoted as $Y$. Our objective is image classification, represented by the model $f : X \rightarrow Y$, which takes an image $x \in X$ as input and predicts its corresponding class label $y \in Y$. In this paper, our focus is on patch detection-based methods. Specifically, we adopt a segmentation model, denoted as $h$, for detecting the adversarial patch in the image. We utilize a binary mask $M \in \{0, 1\}^{W \times H}$ to indicate the true location of the patch.

**Patch Attacks** Patch attacks constructed against detection-based defenses do explicitly optimize to subvert the detector. Rather, these methods attempt to maximize the loss of the downstream classifier, thus implicitly subverting the detector. We find that it is far easier to learn detection based defense schemes which are robust to attacker which do not explicitly subvert the detector. In particular, more potent attacks can be constructed by
incorporating an *evasion-term* into the attackers objective which directly aims to decrease the accuracy of the detector. Attacks generated via this modified objective (discussed in detail below) significantly decrease the efficacy of current detection-based methods. Further, training against attackers which use this modified objective, preserves robustness against attackers which do use standard objectives.

Masked Projected Gradient Descent with infinite steps (Masked PGD-inf) is an advanced adversarial attack strategy specifically designed for patch attacks. It extends the concept of Projected Gradient Descent (PGD) by incorporating a masking mechanism, which allows the attacker to focus on the most vulnerable regions of an image. More specifically, the Masked PGD-inf attack performs gradient updates in a patch region which is likely to maximize the attacker’s objective. When updating the attack at each iteration, a binary mask $M$ is used to zero-out the gradient on the rest of the image. We denote the adversarial image as $X^\delta$, where $\|X^\delta - X\| \leq \epsilon$. Here $\epsilon$ is a scalar parameter that controls the attack strength. The condition $\|X^\delta - X\| \leq \epsilon$ ensures that the perturbations introduced in the adversarial image are within a specified range, allowing for a controlled level of distortion from the original image. Given an input image $X$, its ground truth label $y$, and model $f$ with weights $\theta$ and a defined loss function $\mathcal{L}$, a masked PGD attack is generated by maximizing the loss function in an iteration way:

$$X^\delta_{t+1}[\text{patch}] = C_\epsilon \left( X^\delta_t + \alpha \text{Sign}(\nabla_X \mathcal{L}(X^\delta_t, y, \theta)) \right)[\text{patch}]$$  \hspace{1cm} (7.1)

Note that the clipping function $C_\epsilon$ is utilized to prevent the per-pixel modification from going beyond the $\epsilon$ ball. Here, $\odot$ represents the element-wise multiplication of the mask $M$ with the gradient sign. In the context of patch attacks, the mask $M$ is used to guide the adversarial perturbation towards the most vulnerable regions of an image, which enables the attacker to deceive the classification model more effectively. By iteratively updating the adversarial image using the masked gradients, the attacker can achieve their desired objective while minimizing the overall distortion in the image. While this attack has been shown to be effective in some contexts, it frequently fails to produce strong attacks against detection based defenses. We next introduce a modified objective which is specifically tailored to attack detection-based approaches.
7.4 Method

Our proposed defense falls within the umbrella of detection-based approaches, which work by identifying and neutralizing adversarial examples prior to downstream prediction. In the case of patch-attacks, this amounts to predicting and ablating the adversarial patch. Our approach (outlined in Figure 7.1) is comprised of two key components: a detector which is a segmentation model responsible for predicting regions within a given image which are likely to be adversarial patches, and a semantic-analyzer which is a feature extraction model designed to learn the semantic meaning of the ablated patch regions. Images are first passed through the detector and the predicted patch region is ablated from the image, resulting in a background image. The ablated region is then passed to the semantic-analyzer and embedded into a latent space. Both the background image and the extracted semantics of the ablated patch are then combined together and passed to a downstream classifier. The intuition behind this approach is that the downstream classifier can use the semantic meaning present in the ablated region to better understand the attacker’s strategy.

Before outlining the technical details of this approach, we first discuss the motivation. In typical detection-based defenses for patch attacks [159], the detector is used to predict the location of the adversarial patch, which is then zeroed-out. Attacks against such models can be successful in two primary ways: 1) the patch can avoid being fully detected, thus allowing some adversarially crafted pixels to affect the downstream classifier, and 2) the patch can be crafted to cause the detector to incorrectly ablate an important region of the image, thus decreasing the useful information available in the background image. By embedding the ablated regions in a latent space, the efficacy of attacks corresponding to the two aforementioned cases can be greatly reduced. In the first case only a fraction of the patch is successfully passed to the downstream classifier, and thus a greater strength of pixel perturbations is required throughout the entire patch. These patches carry a high degree of semantic meaning, as outlined later in figure 7.2, which can be easily harnessed by our method to improve predictive efficacy. In the second case, when an important region of the image has been ablated by the detector, this region is still percolated to the downstream classifier under our method.

Note that with a perfect detector (i.e, an oracle) the two methods are equivalent, however perfect detectors are exceedingly rare in practice. As such, our method is uniquely tailored to thrive in situations where the detector’s performance is less than perfect.
7.4.1 Patch Ablation

Following [159], we use a detection module \( h : [0, 1]^{W \times H \times C} \rightarrow \{0, 1\}^{W \times H} \), simply referred to as a detector, which is trained to identify adversarial patches within images. In our proposed pipeline (Figure 7.1) the victim image \( X \in [0, 1]^{W \times H \times C} \) undergoes an initial processing step via \( h \), where \( h(X[i]) \) gives the probability that pixel \( X[i] \) is adversarial (i.e., effected by the patch). Any pixels with a sufficiently high probability is ablated from the image and passed to the patch-branch\(^3\). The remaining pixels, also referred to as the background image, are passed to the background-branch. The output of both branches is then combined and passed to a final decision module. We discuss both branches in further detail, but we first outline the semantic meaning of patches.

7.4.2 Adversarial Semantic Meaning

We advocate for characterizing the attacker’s strategy based on the semantic meaning ingrained in the adversarial patch. We postulate that to effectively change the classification of

\(^3\)In practice we find that passing both the ablated patch and the entire victim image to patch-branch results in a more successful defense
an image $X$ from the correct label $y$ to an incorrect label $y' \neq y$, the adversarial patch must incorporate semantic information associated with label $y'$. We provide further discussion of this phenomenon in the Supplement. The attacker’s strategy (i.e., which label the attack pushes $X$ towards) through the semantic content of the adversarial patch. To do this we use contrastive learning to embed ablated patches into a latent space via the adversarial class $y'$.

**Definition 7.4.1.** (Adversarial Class) For model $g$, image $X$, and patch $\delta$, the adversarial class of $\delta$ is $g(X^\delta)$, i.e., the model’s predicted class on the adversarial example $X^\delta$.

During each epoch of training, we use the the model from the previous epoch to predict the adversarial class of each image. We observe that when constructing untargeted patch attacks, the images from each class $y$ tend to correspond to only a few adversarial classes $y'$ (see Figure 7.3). As such, the observed adversarial class and semantic meaning of the patch, leak information about the true class of a given image. By embedding the semantic meaning of the ablated patches into low dimensional latent space via the adversarial images (as shown in Figure 7.2), the final decision decision model is able to capitalize on this leaked information, thus resulting in greater robustness. As noted later in the experiment section, the low diversity of adversarial classes (for each true class) is still present even after incorporating patch semantic meaning into our method. This indicates that for each true class $y$, there are only a few “good” adversarial classes which images of class $y$ can be flipped to. The narrowness of attacker’s possible strategies is precisely what the patch-branch takes advantage of, and is precisely what is lost when simply zeroing-out the ablated patch.

### 7.4.3 Defense

**Patch Branch** The patch-branch is designed to learn the adversarial semantic meaning, denoted as $z_p$, of the attack. This branch is comprised of model $f_p$ which takes the detected patch with a black background, denoted as $h(X^\delta) \cdot X^\delta$, as input and yields embedding $z_p$. Recall that $h(X[i]) = 0$ indicates that pixel $i$ is predicted as benign. The embedding $z_p$ is created through supervised contrastive learning with loss,

$$
\mathcal{L}_C = \frac{-1}{|P(k)|} \sum_{p \in P(k)} \log \frac{\exp(\langle z_k, z_{i_+} \rangle / \tau)}{\sum_{i_- \in A(k)} \exp(\langle z_k, z_{i_-} \rangle / \tau)}
$$

(7.2)
Figure 7.2: Semantic embedding of ablated patches. Model $g$ gives the adversarial class of each patch (e.g., the first dog is predicted as a cat). Contrastive learning is used to embed ablated patches into the latent space; positive and negative examples are defined via their adversarial class (Definition 7.4.1).

where each input $h(X^δ) \cdot X^δ$ associated with a label, namely the adversarial class of image $X^δ$, i.e., $g(x^δ)$. Here $k$ is the index of an arbitrary example. The set $P(k)$ gives all indices corresponding to positive examples (i.e., those with the same adversarial class), while $A(k)$ gives the indices of negative examples (i.e., those with different adversarial classes). The parameter $\tau \in \mathbb{R}_+$ is the classic temperature parameter. The embedding $z_p$ is then passed to the final decision decision model along with the output of the background branch, the latter of which we discuss next.

**Background Branch**  The background branch utilizes a model, $f_b$, which operates on images after being passes through the detector, which we refer to as *background images*. These background images are generated by applying a mask, derived from the detector $h(X^δ)$, to the initial image $X^δ$, i.e., $X^δ(1 - h(X^δ))$. Background images are then passed to $f_b$, yielding embedding $z_b$. To train the background branch, we use a two layer fully connected network $f'_b$, which takes as input the embedding $z_b$. In the first stage of training we use cross-entropy loss, denoted as $L_B(X, y, f'_b \circ f_b)$. In the second stage of training we use a combination of $L_B$ and the loss of the final decision module to update $f_b$.

Due to the imperfect nature of the detected mask, adversarial pixels may seep into the ablated image and potentially influence on the background branch. To counteract this, we introduce mask augmentation by creating an augmented collection of masks around the originally detected mask using a sampling algorithm $S$. This augmented collection, symbolized as
$S(h(X^δ))$, is applied to ablate the victim image, resulting in a series of ablated images depicted as $X^δ(1 − S(h(X^δ)))$.

**Final Decision Module** The concluding module of the defensive pipeline involves the integration of the outputs of both branches, as graphically represented in Figure 7.1. This fusion process incorporates the adversarial semantic embedding $z_p$ derived from the patch branch and the benign semantic embedding $z_b$ sourced from the background branch. Both embeddings are passed to a final decision module $f_d$ which makes a prediction of the true class of a given image $X$. The learning procedure for this final module is guided by the Cross-Entropy Loss of the prediction of $f_d$ on $X$, denoted as $L_D(X, y, \theta_{f_p}, \theta_{f_b}, \theta_{b})$. As mentioned previously, the final decision module is trained during the second stage of training (after the patch branch and background branch have been warmed up). We next discuss the training paradigm in further detail.

**Training Paradigm** To train our defense, we employ a two-stage process training paradigm which first trains the patch branch, background branch, and detector in isolation, and then trains all three components as well as the final decision module jointly. Each stage consists of a number of rounds (in our experiments we use 10 for the first stage and 100 for the second stage), in which adversarial adversarial examples are generated via untargeted patch attacks against a classifier. These adversarial examples are then used to retrain our defense pipeline. During both stages of training we maintain a memory of adversarial examples generated at previous rounds training. In addition to new adversarial examples created at each round, a random sample is pulled from this memory and used for training.

**First Stage** In the first phase, we train both the background branch and patch branch. To train the background branch we utilize the method proposed in [159]. Adversarial examples are produced by maximizing the loss of the background branch $L_B$. Recall that to train the background branch we use a two layer fully connected network $f'_b$ which takes as input the embedding $z_b$ produced by $f_b$. We initially concentrate on training the models in the background branch $f_b$ and the detector $h$ using adversarial examples generated solely from gradient information derived from the $f'_b \circ f_b$ alone. This done to help improve the stability of $h$. Once the performance of the detector stabilizes, we progress to training with adversarial examples that depend on the gradients from both $f'_b \circ f_b$ and $h$. During this stage, the patch
branch is trained using adversarial labels which are predicted by $f'_b$. We defer the algorithm at stage 1 to the appendix.

**Second Stage** Upon concluding the first stage of training, where the model has gathered insights about the semantic meaning of the background and adversarial patch, we advance to the second stage of training. In this phase, we adopt an end-to-end training approach shown in Algorithm 3 by amalgamating three loss functions - $L_C$, $L_B$, and $L_D$. In the algorithm, we use $g = f \circ h$ represent the full pipeline of AdvSemShield. These loss functions are weighted by their respective coefficients $w_C$, $w_B$, and $w_D$. These are hyperparameters that determine the relative importance or emphasis we wish to place on each loss. The combined loss function is articulated as follows:

$$L = w_C \cdot L_C + w_B \cdot L_B + w_D \cdot L_D$$

When training we initialize $w_C = w_B = w_D = 1$ and linearly decrease $w_C$ and $w_B$ down to .25 over the course of adversarial training.

---

**Algorithm 3** Adversarial Training (Stage 2)

**Data:** Number of training rounds $B$.

```plaintext
begin
    Memory = {}
    for round in $[B]$ do
        $X^\delta = $ UntargetAttack($L_D$, $\theta_f, \theta_f_d, \theta_f_p, X, Y$)
        $g = $ Train($Y, X^\delta, Memory$)
        Add $X^\delta$ to Memory
    end
end
```

---

### 7.4.4 Adversarial Training

The objective is to warm up the different components of the defense pipeline by training them on specific types of adversarial examples. The training process involves two types of attacks: DO Attack and DB Attack.

**BD Attack** BD (background branch and only downstream) attack, proposed by [159], is a specific attack designed for the warm-up phase of the detector in the defense mechanism.
During this stage, the construction of the adversarial examples, denoted as $X^\delta$, is focused on maximizing the loss function $l$, which is defined as follows:

$$y' = f_B((1 - U(h(X^\delta))) * X^\delta) \quad (7.3)$$

In the BD Attack, it is important to note that the binary operation $U$ used in the construction process is not differentiable. As a result, the attack solely relies on the gradient information derived from the downstream classifier in the background branch. This is because, at the early stage of the adversarial training, the gradient information from the detector may not be informative enough since the detector has not yet stabilized. Therefore, focusing on the gradient from the downstream classifier allows for effective construction of adversarial examples during the warm-up phase.

**BF Attack** The BF (background and full) attack is specifically designed to target the entire background branch of the defense mechanism. It is employed after the detector has stabilized during the adversarial training process. In this phase, the non-differentiable binary operation $U$ is replaced with a differentiable proxy function $V$, typically a sigmoid function, as proposed in [159]. The adversarial examples, denoted as $X^\delta$, are now constructed using the differentiable proxy operation $V$, resulting in the following formulation:

$$y' = f_B((1 - V(h(X^\delta))) * X^\delta) \quad (7.4)$$

By adopting the differentiable proxy $V$, the DB Attack enables the use of gradient-based optimization methods during the construction of adversarial examples. This allows for a more representative and refined optimization process, resulting in adversarial examples that are more indicative of the final attack.

**PD Attack** The PD (patch and decision) attack aims to warm up the patch branch by exploiting the gradient solely derived from the patch and decision branches.
Training Paradigm

As mentioned before, we proposed a two-stage adversarial training paradigm for AdvSemShield. Stage 1 of the adversarial training process is focused on warming up the different components of the defense pipeline and training them on specific adversarial examples designed for warmup. This stage consists of two phases, each utilizing different training adversarial examples. We start with introducing the designed adversarial examples in this stage.

In Stage 1, we initialize the downstream classifier \( f_b \) and its counterpart \( f_b' \) as the victim model, which is a classification model trained on clean data. We will use \( f_B \) represent \( \{f_b, f_b'\} \). The first phase involves training the detector \( f_d \) using the DO Attack, where the model is attacked with adversarial examples generated solely from the downstream classifier. Once the detector \( f_d \) becomes stable, we proceed to the second phase, training the detector \( f_d \) and the downstream classifier \( f_b \) with adversarial examples generated by the DB Attack in an online fashion. This process is detailed in Algorithm ??.

Moving to the next phase, we train the patch branch and the decision model on adversarial examples generated by the DB Attack, utilizing a memory-based training paradigm. This approach, outlined in Algorithm Y, enhances the model’s robustness against adversarial attacks by leveraging the semantic understanding of both the adversarial and benign components within the input data.

Through Stage 1, our defense mechanism progressively adapts and learns from different types of adversarial examples, enabling the model to become more resilient to attacks and improve its overall robustness.

The significance of Memory  The utilization of memory in AdvSemShield defense mechanism plays a significant role in capturing and retaining the attacking strategies employed by adversarial examples. By representing the attacking strategies in terms of the semantic meaning of the patch, the strategies become more tractable and understandable. The memory should include all the attacking strategies to ensure that the model fully learns and comprehends these strategies. The memory serves as a repository of adversarial examples, preserving their semantic information and enabling the model to effectively learn and adapt to different attack scenarios. By incorporating the memory into the training process, the model gains a comprehensive understanding of the diverse attacking strategies, enhancing its
ability to detect and mitigate adversarial attacks. The inclusion of memory not only improves the model’s defense against known attacks but also enables it to generalize and adapt to new and unseen attack patterns. This capability is crucial in the ever-evolving landscape of adversarial attacks, where attackers constantly devise new strategies and techniques. Overall, the adoption of memory in our defense mechanism empowers the model to learn the possible attacking strategies and thus enhance its robustness against adversarial attacks.

7.4.5 Evasive Attacks

When crafting adversarial patches, the traditional objective function (Equation 7.1) is to maximize the loss of a given model, in our case \( g = f \circ h \). This formulation, along with optimization schemes such as Masked-PGD, are indifferent to presences of a detector \( h \). Attacks using Equation 7.1 may implicitly attack the detector, but fail to capitalize on its unique role within the predictive pipeline. By modifying the attacker’s objective to explicitly incorporate detector efficacy, significantly stronger attacks can obtained. This modification, discussed next, amounts to incorporating an *evasion term* which decreases the number of modified pixels which are successfully identified by the detector.

Classification loss of model \( g = f \circ h \) on an image \( X \) (with true label \( y \)) is denoted as \( \mathcal{L}_{\text{cls}}(X, y, \theta_f, \theta_h) \); typically \( \mathcal{L}_{\text{cls}} \) refers to the cross-entropy loss. The attacker’s modified loss is thus,

\[
\mathcal{L}_{\text{eva}}(X, y, \theta_f, \theta_h) = (1 - \lambda) \mathcal{L}_{\text{cls}}(X, y, \theta_f, \theta_h) + \lambda \sum_{i=1}^{W \times H} (1 - \max \{\beta, h_{\theta_h}(X)[i]\}) \cdot M[i] \quad (7.5)
\]

where \( \theta_f \) and \( \theta_h \) are the parameters in the \( f \) and \( h \), \( M \) is the true mask of the patch, \( \beta \) is a threshold on detector incorrectness, and \( \lambda \) is the *evasion ratio* which governs the relative importance of attacking each component. The first term focuses on attacking the downstream classifier, aiming to increase the cross-entropy loss between the model’s prediction and the true label. The second term targets the detector directly by reducing the true positive rate with respect to adversarial pixels. Maximizing the right-hand term is equivalent to minimizing the recall of the detector.
We next discuss the role of $\beta$. In essence, $h(X)[i]$ quantifies the prediction confidence of a pixel being classified as adversarial. When the predicted probability of an adversarial pixel being identified as adversarial falls below a given threshold (most commonly chosen to be .5), pixel $X[i]$ is no longer considered adversarial. The term $\beta$ is a hyperparameter that controls the attacker’s desired confidence level for manipulating adversarial pixels. A smaller value of $\beta$ implies a higher level of required confidence when considering an adversarial pixel as successfully passing the detector. Throughout our experiments we found $\beta = .35$ to be most effective.

7.5 Experiment

We conducted experiments on CIFAR10, CIFAR100 and ImageNette, and compare our method with two state-of-the-art baselines, PatchZero [159] and PatchCleanser [155]. PatchZero introduces a two-module pipeline consisting of a detector and classifier. The first module (the detector) utilizes a segmentation model to detect the adversarial patch within an image (similar to our approach). Pixels which are predicted as being adversarial are then zeroed-out, and the resulting image is passed to the second module (the classifier). PatchCleanser, on the other hand, uses a two-round exhaustive masking process which partitions the image into a grid and iteratively zeros-out each element of grid. The final prediction is an aggregate of model predictions for each zeroed-out grid element. We note that PatchCleanser presupposes that the defender has prior knowledge of the adversarial patch’s size, while PatchZero and our method do not.

Attack In designing our experiments, we maintain consistency with the configurations adopted in both PatchZero and PatchCleanser concerning the attack iterations and patch shapes. Specifically, we apply a masked PGD attack with 100 iterations and use a mix of square and rectangular patches of varying sizes. In our experiments we use an $\ell_\infty$ perturbation budget with $\varepsilon = 1$. Additionally we also vary the patch ratio; defined as the patch size divided by the image size. We executed our experiments using the patch ratios employed in PatchZero, namely, 5% and 10%, and additionally include larger patch ratios of 15% to measure the performance of each method under stronger attacks.
To find the attacks, we employed our proposed loss function (Equation 7.5), which incorporates an evasion term specifically utilized for a direct attacks on the detector. We carry out a range of experiments using various evasion ratios $\lambda$, specifically 0.0, 0.25, and 0.5. Importantly, when the evasion ratio is set to 0.0, our loss function reduces to a simple cross-entropy loss function, which recovers the standard patch attack objective, i.e., the setup employed by the the baseline methods. We assess the performance of the baseline methods as well as our proposed method when training against attacks using each of the three evasion terms, as well as testing against attackers using each of the three evasion terms.

<table>
<thead>
<tr>
<th>Methods</th>
<th>EV-0%</th>
<th>EV-25%</th>
<th>EV-50%</th>
<th>EV-0%</th>
<th>EV-25%</th>
<th>EV-50%</th>
<th>EV-0%</th>
<th>EV-25%</th>
<th>EV-50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>PatchZero-0.00%</td>
<td>88%</td>
<td>81.2%</td>
<td>80.3%</td>
<td>79.1%</td>
<td>70.1%</td>
<td>67.1%</td>
<td>77.6%</td>
<td>74.1%</td>
<td>68.1%</td>
</tr>
<tr>
<td>PatchZero-0.25%</td>
<td>88.2%</td>
<td>85%</td>
<td>82%</td>
<td>78.5%</td>
<td>73.1%</td>
<td>73.4%</td>
<td>77.9%</td>
<td>75.2%</td>
<td>70.9%</td>
</tr>
<tr>
<td>PatchZero-0.50%</td>
<td>89.1%</td>
<td>86.2%</td>
<td>85.2%</td>
<td>78.8%</td>
<td>75.1%</td>
<td>73.7%</td>
<td>77.5%</td>
<td>75.2%</td>
<td>73.8%</td>
</tr>
<tr>
<td>PatchCleanser 0.00%</td>
<td>71.2%</td>
<td>–</td>
<td>–</td>
<td>63.8%</td>
<td>–</td>
<td>–</td>
<td>58.2%</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>AdvSemShield-0%</td>
<td>89.1%</td>
<td>85.2%</td>
<td>83.2%</td>
<td>80.2%</td>
<td>76.0%</td>
<td>74.0%</td>
<td>77.6%</td>
<td>76.1%</td>
<td>72%</td>
</tr>
<tr>
<td>AdvSemShield-25%</td>
<td>89.2%</td>
<td>87.9%</td>
<td>85.2%</td>
<td>80.4%</td>
<td>77.1%</td>
<td>75.0%</td>
<td>77.8%</td>
<td>76.5%</td>
<td>75%</td>
</tr>
<tr>
<td>AdvSemShield-50%</td>
<td>88.3%</td>
<td>87.8%</td>
<td>88.8%</td>
<td>80.5%</td>
<td>77.1%</td>
<td>76.0%</td>
<td>76.9%</td>
<td>77.2%</td>
<td>76.1%</td>
</tr>
<tr>
<td>AdvSemShield-0%</td>
<td>70%</td>
<td>63%</td>
<td>56%</td>
<td>66.7%</td>
<td>53%</td>
<td>39.5%</td>
<td>54.2%</td>
<td>59.3%</td>
<td>48.6%</td>
</tr>
<tr>
<td>AdvSemShield-25%</td>
<td>69.2%</td>
<td>66.3%</td>
<td>57%</td>
<td>68.5%</td>
<td>66.2%</td>
<td>65.4%</td>
<td>53.8%</td>
<td>51.2%</td>
<td>49.2%</td>
</tr>
<tr>
<td>AdvSemShield-50%</td>
<td>70.5%</td>
<td>65.2%</td>
<td>60%</td>
<td>68.2%</td>
<td>67.3%</td>
<td>67.2%</td>
<td>54.2%</td>
<td>51.3%</td>
<td>50.2%</td>
</tr>
<tr>
<td>AdvSemShield-0%</td>
<td>71.2%</td>
<td>60.2%</td>
<td>54.8%</td>
<td>66.8%</td>
<td>67.2%</td>
<td>66.1%</td>
<td>59.2%</td>
<td>52.5%</td>
<td>51.3%</td>
</tr>
<tr>
<td>AdvSemShield-25%</td>
<td>72.2%</td>
<td>70.1%</td>
<td>61.2%</td>
<td>70.5%</td>
<td>70.2%</td>
<td>69.2%</td>
<td>59.1%</td>
<td>56.8%</td>
<td>53.3%</td>
</tr>
<tr>
<td>AdvSemShield-50%</td>
<td>73.9%</td>
<td>71.1%</td>
<td>70.2%</td>
<td>72.1%</td>
<td>70.2%</td>
<td>70.3%</td>
<td>59.2%</td>
<td>55.9%</td>
<td>54.1%</td>
</tr>
<tr>
<td>PatchZero-0.00%</td>
<td>94.1%</td>
<td>86.3%</td>
<td>80%</td>
<td>90.6%</td>
<td>89.2%</td>
<td>74.0%</td>
<td>86.2%</td>
<td>71.3%</td>
<td>65.3%</td>
</tr>
<tr>
<td>PatchZero-0.25%</td>
<td>95.1%</td>
<td>88.2%</td>
<td>83%</td>
<td>90.9%</td>
<td>87.1%</td>
<td>78.1%</td>
<td>88.1%</td>
<td>80.3%</td>
<td>73.7%</td>
</tr>
<tr>
<td>PatchZero-0.50%</td>
<td>94.9%</td>
<td>87.9%</td>
<td>85%</td>
<td>90.8%</td>
<td>87.3%</td>
<td>82.4%</td>
<td>89.5%</td>
<td>81.2%</td>
<td>78.9%</td>
</tr>
<tr>
<td>PatchCleanser 0.00%</td>
<td>80.2%</td>
<td>–</td>
<td>–</td>
<td>45.1%</td>
<td>–</td>
<td>–</td>
<td>32.1%</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>AdvSemShield-0%</td>
<td>94.8%</td>
<td>88.2%</td>
<td>82%</td>
<td>90%</td>
<td>83.2%</td>
<td>78.1%</td>
<td>88.1%</td>
<td>74.2%</td>
<td>72.1%</td>
</tr>
<tr>
<td>AdvSemShield-25%</td>
<td>95.1%</td>
<td>89.1%</td>
<td>85%</td>
<td>89.5%</td>
<td>85.3%</td>
<td>83%</td>
<td>88.2%</td>
<td>83%</td>
<td>79.7%</td>
</tr>
<tr>
<td>AdvSemShield-50%</td>
<td>95.2%</td>
<td>89.9%</td>
<td>88.5%</td>
<td>91.2%</td>
<td>89.2%</td>
<td>86.5%</td>
<td>87.5%</td>
<td>84.9%</td>
<td>83.1%</td>
</tr>
</tbody>
</table>

Table 7.1: Model robustness across different datasets, patch ratios, and evasion terms.

### 7.5.1 Results

**Robustness** Table 7.1 shows the performance of all three approaches across each dataset. In the table, the term PatchZero—0.25% denotes that we’re employing the PatchZero method, and training is carried out on adversarial examples where the evasion ratio is set at 25%.

<table>
<thead>
<tr>
<th>Methods</th>
<th>EV-0%</th>
<th>EV-25%</th>
<th>EV-50%</th>
<th>EV-0%</th>
<th>EV-25%</th>
<th>EV-50%</th>
<th>EV-0%</th>
<th>EV-25%</th>
<th>EV-50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>PatchZero</td>
<td>88.9%</td>
<td>74.8%</td>
<td>95.2%</td>
<td>86.2%</td>
<td>74%</td>
<td>83.3%</td>
<td>85.2%</td>
<td>73.1%</td>
<td>85.2%</td>
</tr>
<tr>
<td>PatchCleanser</td>
<td>88.4%</td>
<td>74.9%</td>
<td>96%</td>
<td>86.1%</td>
<td>75.3%</td>
<td>94.2%</td>
<td>88.4%</td>
<td>72.9%</td>
<td>91.8%</td>
</tr>
<tr>
<td>AdvSemShield</td>
<td>89.3%</td>
<td>74.7%</td>
<td>95.9%</td>
<td>86.9%</td>
<td>76.2%</td>
<td>94.8%</td>
<td>87.2%</td>
<td>74.1%</td>
<td>92.1%</td>
</tr>
</tbody>
</table>

Table 7.2: Clean accuracy
Table 7.3: Accuracy of a 2-layer fully connected network which uses the latent space of the patch-branch to predict the adversarial label (Definition 7.4.1.)

<table>
<thead>
<tr>
<th>Methods</th>
<th>CIFAR10</th>
<th>CIFAR100</th>
<th>ImageNette</th>
<th>CIFAR10</th>
<th>CIFAR100</th>
<th>ImageNette</th>
<th>CIFAR10</th>
<th>CIFAR100</th>
<th>ImageNette</th>
</tr>
</thead>
<tbody>
<tr>
<td>EV-0.0</td>
<td>74.4%</td>
<td>52.1%</td>
<td>80.1%</td>
<td>73.1%</td>
<td>51.2%</td>
<td>81.1%</td>
<td>57%</td>
<td>51.1%</td>
<td>79.1%</td>
</tr>
<tr>
<td>EV-0.25</td>
<td>72.1%</td>
<td>51.1%</td>
<td>81.2%</td>
<td>69.1%</td>
<td>48.1%</td>
<td>82.1%</td>
<td>68.2%</td>
<td>47.3%</td>
<td>79.1%</td>
</tr>
<tr>
<td>EV-0.5</td>
<td>70.1%</td>
<td>49.9%</td>
<td>83.1%</td>
<td>70.2%</td>
<td>50.3%</td>
<td>79.8%</td>
<td>67.3%</td>
<td>49.1%</td>
<td>79.8%</td>
</tr>
</tbody>
</table>

The same interpretation applies to AdvSemShield. Note that for PatchCleanser the evasion attack is ill defined as this approach does not use a detector. PatchZero and AdvSemShield employ segmentation models to locate adversarial patches, contrasting with PatchCleanser’s exhaustive algorithm. As we see in the table using a learnable segmentation model leads to greater levels of robustness. Further we see that our method consistently our performs both both baselines, indicating that the incorporation of adversarial semantic meaning allows stronger defenses. The performance of these methods is largely influenced by the efficacy of the detector. With an evasion ratio of 0.0, the attacker finds it challenging to avoid detection, particularly for the CIFAR-10 and ImageNette datasets, and thus crafts weaker attacks. In these instances, the segmentation-based detectors outperforms the detection algorithm from PatchCleanser (results for detection success are shown in the Supplement). Both PatchZero and AdvSemShield exhibit superior defenses against patch attacks. These methods perform comparably, as illustrated in the table, as the best defense strategy is to precisely identify and remove the adversarial patch from the image. Similarly, we observe that the inclusion of a nonzero evasion ratio also helps to improve model robustness (even against non-evasive attacks).

We thus see that it is essential for defenses that utilize learnable detectors to train on attacks using a range of evasion ratios with the proposed loss function (see Equation 7.5). This diverse training enables the defense mechanism to handle a variety of evasion strategies from attackers. Remarkably, our method, AdvSemShield-0.0, demonstrates considerable resilience against different evasion ratios during testing, outperforming the baseline. This highlights our approach’s ability to adapt to various attack scenarios, significantly enhancing model robustness.

By incorporating adversarial semantic meaning and merging the patch and background branches, our defense mechanism is able to achieve greater robustness to adversarial attacks. AdvSemShield’s superiority over PatchZero and PatchCleanser stems from its ability to learn and apply the semantic meaning of the patch (we latter show results for the predictive efficacy
of the embeddings of the patch branch). Ultimately, this diminishes the semantic meaning of the patch thus weakening the attack. As a result, adversarial training between the attacker and AdvSemShield converges towards a weaker optimal attack. In a similar vain, we show in Figure 7.3 that after retraining, the distribution of adversarial labels has greater diversity under out method than under PatchZero. This indicates that the attacker is required to find more diverse (potentially more suboptimal) strategies when crafting attacks against our defense. In Table 7.2, we show the accuracy of each approach on clean images. We observe that all three methods have nearly identical performance on clean data, indicating that our superior robustness has not come at the cost of predictive efficacy on benign examples.

$$TPR = \frac{\sum_i U(h(X[i]) \cdot M[i]}{\sum_i M[i]}$$

Here $X[i]$ represents pixel $i$ in image $X$, and $U(h(X)) \in \{0, 1\}^{W \times H}$ is the predicted binary mask, and $M$ represents the ground truth mask of the given image $X$.

<table>
<thead>
<tr>
<th>Methods</th>
<th>EV-0%</th>
<th>EV-25%</th>
<th>EV-50%</th>
<th>EV-0%</th>
<th>EV-25%</th>
<th>EV-50%</th>
<th>EV-0%</th>
<th>EV-25%</th>
<th>EV-50%</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CIFAR10</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PatchZero-0.00%</td>
<td>94.1%</td>
<td>89.2%</td>
<td>66.3%</td>
<td>97%</td>
<td>90.1%</td>
<td>67.5%</td>
<td>98.36%</td>
<td>87.1%</td>
<td>83.5%</td>
</tr>
<tr>
<td>PatchZero-0.25%</td>
<td>93.9%</td>
<td>91.1%</td>
<td>71.1%</td>
<td>97.0%</td>
<td>92.1%</td>
<td>74.4%</td>
<td>98.2%</td>
<td>95.2%</td>
<td>87.9%</td>
</tr>
<tr>
<td>PatchZero-0.50%</td>
<td>94.1%</td>
<td>91.6%</td>
<td>75.2%</td>
<td>97.12%</td>
<td>92.0%</td>
<td>79.7%</td>
<td>98.5%</td>
<td>94.5%</td>
<td>88%</td>
</tr>
<tr>
<td>PatchCleaner</td>
<td>50.2%</td>
<td>-</td>
<td>-</td>
<td>42.8%</td>
<td>-</td>
<td>-</td>
<td>40.2%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>AdvSemShield-0%</td>
<td>94.1%</td>
<td>90.2%</td>
<td>69.2%</td>
<td>97.2%</td>
<td>92.0%</td>
<td>67.0%</td>
<td>98.0%</td>
<td>87.9%</td>
<td>84.1%</td>
</tr>
<tr>
<td>AdvSemShield-25%</td>
<td>94.4%</td>
<td>91.0%</td>
<td>71.2%</td>
<td>98.0%</td>
<td>93.0%</td>
<td>76.0%</td>
<td>98.8%</td>
<td>96.5%</td>
<td>87.1%</td>
</tr>
<tr>
<td>AdvSemShield-50%</td>
<td>94.8%</td>
<td>91.5%</td>
<td>76.1%</td>
<td>97.9%</td>
<td>92.5%</td>
<td>80.1%</td>
<td>98.9%</td>
<td>96.7%</td>
<td>89.1%</td>
</tr>
</tbody>
</table>

| CIFAR100         |       |        |        |       |        |        |       |        |        |
| PatchZero-0.00%  | 90.1% | 40%    | 32.1%  | 91.1% | 53%    | 35.1%  | 99.5% | 92.3%  | 86.6%  |
| PatchZero-0.25%  | 90.2% | 64.3%  | 59.1%  | 91.5% | 68.9%  | 61.3%  | 99.8% | 93.0%  | 87.2%  |
| PatchZero-0.50%  | 90.5% | 64.1%  | 61.2%  | 91.2% | 68.3%  | 60.2%  | 99.3% | 93.3%  | 88.5%  |
| PatchCleaner     | 55%   | -      | -      | 50.1% | -      | -      | 45.1% | -      | -      |
| AdvSemShield-0%  | 90.2% | 44.2%  | 30.8%  | 91.2% | 50.2%  | 36.1%  | 99.2% | 92.2%  | 85.3%  |
| AdvSemShield-25% | 91.0% | 70.1%  | 61.2%  | 91.5% | 65.0%  | 62.2%  | 99.1% | 93%    | 88.3%  |
| AdvSemShield-50% | 90.1% | 71.1%  | 73.1%  | 92.0% | 68.1%  | 63.7%  | 99%  | 93.1%  | 89.1%  |

| ImageNet         |       |        |        |       |        |        |       |        |        |
| PatchZero-0.00%  | 94.1% | 86.3%  | 80%    | 96.6% | 51.3%  | 47.8%  | 96.9% | 51.1%  | 55.9%  |
| PatchZero-0.25%  | 95.1% | 88.2%  | 83%    | 96.9% | 64.2%  | 59.1%  | 97.6% | 69.2%  | 60.8%  |
| PatchZero-0.50%  | 94.9% | 87.9%  | 85%    | 98.0% | 63.9%  | 63.2%  | 97.0% | 69%    | 65.2%  |
| PatchCleaner     | 65.2% | -      | -      | 60.8% | -      | -      | 55.6% | -      | -      |
| AdvSemShield-0%  | 94.0% | 85.2%  | 82%    | 96.5% | 52.2%  | 48.1%  | 97.1% | 52.2%  | 54.1%  |
| AdvSemShield-25% | 94.2% | 88.1%  | 83%    | 95.9% | 66.1%  | 59.1%  | 97.2% | 70%    | 61.7%  |
| AdvSemShield-50% | 94.1% | 88%    | 86.1%  | 96%   | 65.4%  | 64%    | 97.5% | 69.2%  | 64.9%  |

Table 7.4: True Positive Rate of the detector used in each method. For PatchZero and AdvSemShield the False Positive Rate of the detector is less than 0.01 in all cases.

Table 7.4 presents an analysis of the detector’s efficacy across different datasets, patch ratios, and evasion terms. It provides a comprehensive overview of the detection performance of the
three methods. This table corresponds to Table 7.1 in the main body. Upon examining the table, we observe that both AdvSemShield and PatchZero in the detector demonstrate superior TPR performance compared to the detection algorithm proposed by PatchCleanser. Additionally, the effectiveness of detecting adversarial examples is similar for both AdvSemShield and PatchZero across a variety of datasets. For larger evasion ratios, the FPR of both methods remains low $< 0.01$. This stems from the fact that the evasion attack optimizes for detector FNR and is thus unlikely to increase FPR. We experimented with optimizing for both FPR and FNR, but found that the inclusion of FPR decreased the strength of generated attacks. This likely stems from the fact that each adversarial pixel which passes the detector has greater co

Efficacy of the Patch branch Lastly we examine the efficacy of the embeddings produced by the patch branch. In Table 7.3 we see the efficacy of a two layer neural network trained on the embedding of the patch branch $z_p$. These results are reported as the accuracy of predicting the adversarial class of each ablated patch. We see that across all the three datasets the patch branch produces useful embedding, in that these embedding can be used by a simple model to accurately predict the adversarial class corresponding to each image. We note this is particularly true for CIFAR10 and ImageNette (which have 10 classes) compared to CIFAR100 (which has 100). As the number of classes increases, the adversary has a broader range of possible strategies, and thus the strategies become more difficult to predict.
Chapter 8

Certified Robust Control under Adversarial Perturbations

8.1 Introduction

Traditional autonomous systems rely on highly reliable control algorithms and high quality sensory information to perform relatively narrowly defined tasks, such as vehicle autopilot [32] and robotic assembly line control [175, 23]. Increasingly, however, the notion of autonomy has broadened to involve complex behavior in broader domains, such as autonomous driving, where sensory measurements are high-dimensional, obtained using a camera and/or LiDAR [145, 82]. In such domains, modern algorithmic approaches for computer vision have become critical as a means to compress complex sensory data into interpretable information that can subsequently be used in control. In particular, transformative advances in the use of deep neural networks for common vision tasks such as image classification and object detection have enabled practical advances in problems such as autonomous driving [15].

Despite considerable advances, however, neural network models that are highly effective in visual prediction tasks are nevertheless also highly susceptible to small (often imperceptible) adversarial perturbations to the same inputs [15]. In turn, extensive literature has emerged to investigate approaches for robust machine learning [26, 123], A common goal of certified robustness is prediction invariance: that is, what is the maximum that an input can be adversarially perturbed without changing the prediction [123, 26]? As prediction invariance is only sensible in classification, its natural regression counterpart certifies a prediction interval for a specified bound on the magnitude of the adversarial perturbation [18].

However, predictions are typically a means to control, and mistakes in predictions are significant because they can result in catastrophic mistakes in control, such as a crash of an
autonomous car. As such, disembodied certification on prediction properties is inherently limited. For example, invariance is often too strict since alternative predictions may have little impact on system properties, such as safety and stability. It is clearly crucial to couple certified robustness of predictions with control in a way that enables us to certify the natural robustness properties of controllers, such as stability.

We propose a simple approach for combining robustness certification of prediction (either classification or regression) with control by making use of robust control algorithms that leverage uncertainty sets about time-invariant dynamic system parameters as input. This, coupled with a notion of class-conditional safety sets, enables us to obtain end-to-end certificates of controller robustness under adversarial perturbations to raw high-dimensional sensory inputs. We then instantiate our approach in the context of vehicle lateral dynamics, obtaining a control algorithm that yields a robust controller that is composed of interval-based prediction certificates. Finally, we extensively evaluate the proposed approach for end-to-end certified robustness of composition of vision and control, demonstrating the value of the certificates.

8.2 Additional Related Work

The problem of adversarial perturbations to inputs has now been studied, particularly in the context of computer vision [49, 14, 164, 40]. Moreover, a number of recent efforts have been devoted to developing techniques to improve the robustness of machine learning to adversarial perturbations [144, 56, 55, 122], with many such approaches aiming to formally certify robustness [18, 124]. Our work blends certified robustness of perceptual reasoning with robust adaptive control. Adaptive control, which adapts a controlled system to an uncertain environment by adjusting uncertain parameters, has been studied for a few decades [59, 126]. With the advance of machine learning, recent works expand adaptive control to learning-based control, which can learn more complex and higher dimensional functions [3, 41, 68]. Since the learning-based control cares about system stability and safety, it is often called a safe-learning. The common idea is to defer exploring potentially unsafe regions until after getting sufficient data. Due to this assumption, the system with learning-based controls is in danger of failure when applied to autonomous vehicles that operate in dynamically changing environments, where they cannot choose mild and safe environments to explore. As
a result, when they begin to learn dynamic systems in uncertain environments, they may already lose control, and it is too late to restore controllability. In terms of learning-based control, the current paper addresses the problem of those controllers’ reactive nature with respect to environmental changes by incorporating vision. In particular, the proposed control system predicts an uncertain environment from look-head information and adapts to this environment in advance.

8.3 Preliminaries

Consider the dynamical system of the following form:

\[
\dot{s}(t) = G(y, s(t), \pi(t), w, \theta, \sigma(t)) \quad (8.1a)
\]
\[
o(t) = c^T s(t) \quad (8.1b)
\]

where \(s(t)\) is true system state at time \(t\), \(\pi(t)\) is controller, \(y \in \mathbb{R}^m\) is a vector of real-valued parameters that influence system dynamics, \(c\) is the known output matrix, \(o(t)\) are measurements, and \(w, \theta, \) and \(\sigma(t)\) are unknown input gain, state-dependent uncertainty, and time-varying uncertainty, respectively. All of the uncertainties can also depend on \(y\). We will discuss this later. A common goal in robust adaptive control, such as \(L_1\) adaptive control, is to design a controller \(\pi(t)\) which yields stability in the limit and also guarantees bounded transient tracking error. We formalize this goal as follows. Let \(\pi_{ref}\) be the reference controller and \(s_{ref}\) the reference state, which correspond to system behavior when uncertainty is perfectly tracked during uncertainty estimation (this will be clear below when we instantiate our setting in the concrete lateral vehicle control setting). Additionally, let \(\pi_{des}\) and \(s_{des}\) be the design controller and state, respectively which are associated with ideal system behavior (i.e., where error is 0 for all \(t\)). We now formalize our particular meaning of robust control here.

**Definition 8.3.1.** A controller \(\pi(t)\) is robust if there exist positive constants \(c_1\) and \(c_2\) such that (1) \(\|s_{des} - s(t)\|_\infty \leq c_1 \) & \(\|\pi_{des} - \pi(t)\|_\infty \leq c_2\) for all \(t\), and (2) \(\lim_{t \to \infty} \|s_{ref}(t) - s(t)\|_\infty = 0\) & \(\lim_{t \to \infty} \|\pi_{ref}(t) - \pi(t)\|_\infty = 0\).

Our central focus is the case where uncertainty in the dynamics stems predominantly from uncertainty about \(y\). In particular, below we will consider an autonomous driving setting in
which \( y \) corresponds to friction (more precisely, cornering stiffness of the vehicle that results from it), and we estimate \( y \) by first obtaining a high-dimensional visual input \( x \) (e.g., a camera frame) through the use of a deep neural network \( f(x) \). Thus, the dynamical system is a composition of predictions mapping raw sensory inputs \( x \) into parameters of system dynamics, state, and controller. In particular, the central source of uncertainty that we are concerned about are adversarial perturbations to the input image \( x \), denoted by \( \delta \), where \( f(x + \delta) \) is substantively different from \( f(x) \). We consider two prediction cases: classification and regression.

Common efforts on certifying robustness of predictions to adversarial perturbations is focused on prediction invariance [18, 26]. When we couple predictions \( f(x) \) and dynamics and control in Equation (8.1), however, not all errors are equally consequential (some may destabilize the system, whereas others will not significantly change stability), and some prediction errors may seem small in absolute terms, but can result in severe safety violations. Our goal is to enable certification of robust control to adversarial perturbations to raw sensory inputs \( x \) of the system described above composed of predictions \( f(x) \) and dynamics in Equation (8.1).

It will be useful below to take advantage of the transparent semantics of parameters \( y \) in the context of classification-based predictions \( f(x) \) to define for each label \( l \in L \) a safe set of labels \( S(l) \). For example, if the true label is that the weather is sunny, predicting that it is rainy is “safe” in the sense that it would cause the controller to only be more conservative. On the other hand, predicting that the weather is sunny on a rainy day potentially leads to unsafe behavior.

### 8.4 Certifying Robustness of Control to Adversarial Input Perturbations

We now present our approach for certifying robustness of control of dynamical systems described in Equation (8.1) in which a function \( f(x) \) (e.g., a deep neural network) uses raw perceptual inputs \( x \) to predict parameters \( y \) of system dynamics. We focus attention on adversarial perturbations \( \delta \) with bounded \( \ell_2 \) norm. In particular, we will build on the techniques of randomized smoothing [26] and percentile smoothing [18] in order to obtain bounds on \( \|\delta\|_2 \) that guarantee that the controller is robust as formalized in Definition 8.3.1.
to arbitrary adversarial perturbations within these bounds. We first consider the classification and subsequently the regression variants of the prediction problem.

**Classification Settings**  Consider a classifier $f(x)$ that outputs a label $l$ which is then mapped to a set $Y$ of possible values for system parameters $y$, and recall that for each $l \in L$, $S(l)$ is a set of safe predictions. We now construct a *smoothed* classifier $g(x)$ as follows. Let $\gamma$ be a random variable distributed according to a zero-mean isotropic Gaussian distribution $\mathcal{N}(0, v^2 I)$, where $I$ is the identity matrix and $v^2$ the variance (which we would specify exogenously to balance the tradeoff between performance and robustness). Then $g(x) = \nu \mathbb{P}\{f(x + \gamma) = l'\}$ is the smoothed counterpart of $f(x)$ for each input $x$, where the probability is with respect to $\gamma$. In practice, we estimate $g(x)$ by Monte-Carlo sampling [26]. The next result is a direct adaptation of prior results certifying robustness of $g(x)$ to allow us to consider safe sets of labels $S(l)$. Proposition 8.4.1 gives the robust function $g(x)$ a certificate in terms of the strength of the adversarial perturbation. If the additive corruption to the input is within this certificate, the smoothed function guarantees its prediction of this adversarial input is within the safe set.

**Proposition 8.4.1.** Let $a =_{a \in L} g(x)$ and $b =_{b \in L \setminus S(a)} g(x)$. Then $g(x + \delta) \subseteq S(a)$, for all $\delta$ such that $\delta^2 \leq \tau$, where $\tau = \frac{v}{2} (\Phi^{-1}(P_a) - \Phi^{-1}(P_b))$, and $P_a = \mathbb{P}(f(x + \gamma) = a), P_b = \mathbb{P}(f(x + \gamma) = b)$.

**Proof.** We defer the proof to the full version of the paper. \hfill \square

We use Proposition 8.4.1 combined with conventional robust control to provide the end-to-end robustness guarantee. First, we define what we mean by a robust control algorithm.

**Definition 8.4.2.** Suppose that $\mathcal{A}$ is a control algorithm that takes as input a specification (8.1) of a dynamical system and a set $Y$ such that the true system parameters $y \in Y$. We say that $\mathcal{A}$ is robust if it returns a robust policy $\pi(t)$. We use $\mathcal{A}(Y)$ to explicitly indicate that $\mathcal{A}$ takes the set $Y$ as input.

We will discuss a particular robust adaptive control method for vehicle lateral dynamics. The next key result follows by the definition of a robust control algorithm and Proposition 8.4.1.
Theorem 8.4.3 (Classification Setting). Suppose that $y \in \zeta(g(x))$ (i.e., $g(x)$ produces a prediction, and maps $\zeta$ to system parameters) and let $A$ be a robust control algorithm. Then $A(\zeta(g(x+\delta)))$ is robust for any $\delta$ such that $\delta_2 \leq \tau$, where $\tau$ is as defined in Proposition 8.4.1.

In the adversarial setting, if the malicious corruption in the environment is within the certified radius, the predicted $y$ system dynamics parameters from the robust model $g$ with input image $x$ is within the safe range. The control algorithm $A$ thus returns a robust policy.

Regression Settings  Consider now a case in which $f(x)$ is a regression. Since we can treat each coordinate of $y$ independently, we will assume that $y$ is a scalar (i.e., a single parameter of system dynamics). Let $\gamma$ again be zero-mean isotropic Gaussian noise as above, and define

$$h_p(x) = \inf \{ y \in \mathbb{R} | \Pr(f(x + \gamma) \leq y) \geq p \}. \quad (8.2)$$

At the high level, $h_p(x)$ is the $p$th percentile of the distribution of values of $y = f(x + \gamma)$. We will use the median of this distribution as our smoothed regression prediction, which we denote by $h^*(x) \equiv h_{0.5}(x)$. We make use of the following result due to Chiang et al. [18]:

Proposition 8.4.4 ([18]). For any $\epsilon$ and $\|\delta\|_2 \leq \epsilon$,

$$h_p(x) \leq h_p(x + \delta) \leq h_{\overline{p}}(x), \quad (8.3)$$

where $\overline{p} := \Phi(\Phi^{-1}(p) - \frac{\epsilon}{\sqrt{2}})$ and $\overline{p} := \Phi(\Phi^{-1}(p) + \frac{\epsilon}{\sqrt{2}})$.

In particular, if $\underline{p} := \Phi(\Phi^{-1}(0.5) - \frac{\epsilon}{\sqrt{2}})$ and $\overline{p} := \Phi(\Phi^{-1}(0.5) + \frac{\epsilon}{\sqrt{2}})$, then $h^*(x + \delta) \in [h_{\underline{p}}(x), h_{\overline{p}}(x)]$ for any adversarial perturbation $\delta$ with $\|\delta\|_2 \leq \epsilon$. We can again make use of this to obtain the following key result:

Theorem 8.4.5 (Regression Setting). Suppose that $|h^*(x) - y| \leq \beta$, where $y$ is the true parameter value given input $x$. Let $y = \min \{ h_{\underline{p}}(x), h^*(x) - \beta \}$ and $\overline{y} = \max \{ h_{\overline{p}}(x), h^*(x) + \beta \}$, where $\underline{p} := \Phi(\Phi^{-1}(0.5) - \frac{\epsilon}{\sqrt{2}})$ and $\overline{p} := \Phi(\Phi^{-1}(0.5) + \frac{\epsilon}{\sqrt{2}})$. Then for any $\epsilon > 0$, $A([y, \overline{y}])$ is robust for any $\delta$ with $\|\delta\|_2 \leq \epsilon$.

The result follows since the conditions in the theorem ensure that the true parameters $y \in [y, \overline{y}]$. What is particularly surprising is that this holds true for an arbitrary $\epsilon$—that is,
adversarial perturbations of arbitrary magnitude. The reason that arbitrary perturbations cannot destabilize the system $A(\zeta(g(x+\delta)))$ is that although the perception of the environment can be maliciously modified, the robust perception model $g$ still yields a certified interval that contains the true system dynamics parameter at the current state. The downstream control algorithm $A$ thus always returns a stable control policy. While this is so, higher levels of $\epsilon$ entail looser intervals $[y, \bar{y}]$, which in turn means degraded controller performance accordingly (e.g., the vehicle stops).

8.5 Certified Robust Vehicle Control

8.5.1 Vehicle Lateral Dynamics

The current section describes the model for (8.1) on which the paper relies and the control goal.

**Dynamic model** We use the bicycle model [116] to model the vehicle longitudinal dynamic for lateral position $q^y$ and yaw angle $q^\psi$. Given longitudinal velocity $V$, desired lateral position $q^{y,\text{des}}$, and desired yaw angle $q^{\psi,\text{des}}$, the differential equation of the bicycle model can be expressed as the error dynamics ((2.45) in [116]):

$$\dot{s} = As + b \pi + g \dot{q}^{\psi,\text{des}}, \quad (8.4)$$

where $s = [s_1, \dot{s}_1, s_2, \dot{s}_2]^\top$, $s_1 = q^y - q^{y,\text{des}}$ and $s_2 = q^\psi - q^{\psi,\text{des}}$ are the error states, $\dot{q}^{\psi,\text{des}} = \frac{V}{R}$ is the rate of the desired yaw angle, and $R$ is the radius of the road. Control input $u = d$
represents front steering angle. The system matrices are

\[
A = \begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & -2 \frac{C_f + C_r}{mV} & 2 \frac{C_f + C_r}{m} & 0 \\
0 & 0 & 0 & 1 \\
0 & -2 \frac{C_f \ell_f - C_r \ell_r}{I_z V} & 2 \frac{C_f \ell_f - C_r \ell_r}{I_z} & -2 \frac{C_f \ell_f^2 + C_r \ell_r^2}{I_z V}
\end{bmatrix}
\]

\[
b = \begin{bmatrix}
0 \\
2 \frac{C_f}{m} \\
0 \\
2 \frac{C_f \ell_f}{I_z}
\end{bmatrix}
\]

\[
g = \begin{bmatrix}
0 \\
-2 \frac{C_f \ell_f - C_r \ell_r}{mV} & 0 & -2 \frac{C_f \ell_f^2 + C_r \ell_r^2}{I_z V}
\end{bmatrix}
\]

where \( m \) is the vehicle mass and \( I_z \) is the yaw moment of inertia, \( \ell_f, \ell_r \) are the front/rear tire distance from the center of gravity, and \( C_f, C_r \) are front/rear cornering stiffness. Matrices \( A \) and \( g \) depend on velocity \( V \), and \( A, b, \) and \( g \) depend on cornering stiffnesses \( C_f \) and \( C_r \). The cornering stiffness \( C_f \) and \( C_r \) have a linear relation \( F_f = C_f \nu \) with respect to the lateral force \( F_f \) for a small sliding angle \( \nu \).

**Uncertainty model**  The cornering stiffnesses \( C_f \) and \( C_r \) are the road parameters where the vehicle is driving. Thus it is reasonable to assume that they are time-varying and unknown in advance. Consequently, we obtain them by predicting road friction from raw sensory inputs \( x \). However, we aim to ensure the robustness of control to adversarial perturbations \( \delta \) to raw inputs \( x \), and the resulting prediction error induces uncertainty in the dynamic model (8.4). Henceforth, to simplify discussion we assume \( C_f = C_r \equiv C \).

**Control objective**  We aim to stabilize the error state \( s \) in (8.4) so that the vehicle can keep the desired center lane despite adversarial perturbations to raw sensory inputs \( x \).

### 8.5.2 \( L_1 \) Adaptive Control Design

The key control challenge is that the system matrices in the lateral error dynamic (8.4) are unknown because they are subject to unknown and time-varying cornering stiffness \( C \). Instead, we observe raw camera input \( x \) that provides indirect and potentially noisy information about \( C \), using two approaches for predicting \( C \): 1) classification and 2) regression. In the
classification variant, we have a model $f(x)$ that predicts discrete properties of the scene captured by a camera, such as weather or road surface type. In addition, each predicted class $l$ is associated with a cornering stiffness (friction) interval $[C_l, \overline{C}_l]$. In regression, our model $f(x)$ directly predicts road cornering stiffness, i.e., $C = f(x)$.

To induce provable robustness to adversarial perturbations, rather than using $f(x)$ directly for predictions, we apply randomized smoothing in the case of classification, obtaining a smoothed function $f(x)$, or median smoothing in the case of regression, obtaining $h^*(x)$. As discussed in Section 8.4, these can be associated with either a safe prediction set $S(l)$ and associated certification radius for classification or a certified interval for $h^*(x)$. In either case, the procedure yields an uncertainty interval $[\underline{C}, \overline{C}]$ for cornering stiffness.

To deal with the control problem in the presence of uncertainty about cornering stiffness, we will utilize $L_1$ adaptive controller [59] that can rapidly compensate the impact of uncertainties within the designed filter bandwidth of it, and guarantee transient tracking error even when unknown parameters are changing. In what follows, we will explain controller design procedure in detail.

**Nominal Model**

The first step is to transform the model (8.4) into a nominal model, where we will move any uncertainties out of the system matrices. As a result, the nominal system matrices are known, and have desired system properties including stability. We will then design the $L_1$ adaptive controller which forces the system (8.4) to behave like the nominal model by canceling out the uncertainties.

Recall that our prediction models (either classification or regression) yield an uncertainty interval for cornering stiffness. The key assumption we make about this interval is that it includes both the true and predicted (nominal) values:

**Assumption 8.5.1.** The control algorithm takes as input an interval $[\underline{C}, \overline{C}]$ such that $C, \hat{C} \in [\underline{C}, \overline{C}]$, where $C$ is the true and $\hat{C}$ nominal cornering stiffness.
If we take \( \pi(t) = -k_m s(t) + \pi_{ad}(t) \), the system (8.4) can then be transformed into the following nominal model:

\[
\begin{align*}
\dot{s}(t) &= A_m s(t) + b_m (w \pi_{ad}(t) + \theta^\top s(t) + \sigma(t)) \\
\sigma(t) &= c^\top s(t) \quad x(0) = x_0,
\end{align*}
\] (8.5)

where \( A_m = A(\hat{C}, V) - k_m s \) is Hurwitz, and \( b_m = b(\hat{C}) \). The gain \( k_m \) will be determined later. The unknown parameters \( w, \theta, \) and \( \sigma(t) \) are induced by the uncertainty about cornering stiffness \( C \).

**Adaptive Controller Design**

In order to obtain both system stability and bounded transient error, we design an adaptive controller \( \pi_{ad}(t) \) in (8.6) that aims to cancel out the residual uncertainty \( w \pi_{ad}(t) + \theta^\top s(t) + \sigma(t) = 0 \) stemming from uncertainty about \( C \). Adaptive controller \( \pi_{ad}(t) \) consists of state predictor, adaptation law, and low-pass filter as described below. The state predictor is designed using the known parts of the dynamic system in (8.6) and the states of uncertainties:

\[
\begin{align*}
\hat{s}(t) &= A_m \hat{s}(t) + b_m (\hat{w}(t) \pi_{ad}(t) + \hat{\theta}^\top s(t) + \hat{\sigma}(t)) \\
\hat{y}(t) &= c^\top \hat{s}(t), \quad \hat{s}(0) = \hat{s}_0.
\end{align*}
\]

We design the adaptation law to estimate uncertainties:

\[
\begin{align*}
\dot{\hat{w}}(t) &= \Gamma \text{Proj}(\hat{w}(t), -\hat{s}^\top(t) P b_m \pi_{ad}(t)) \quad \hat{w}(0) = \hat{w}_0 \quad (8.7) \\
\dot{\hat{\theta}}(t) &= \Gamma \text{Proj}(\hat{\theta}(t), -\hat{s}^\top(t) P b_m s(t)) \quad \hat{\theta}(0) = \hat{\theta}_0 \quad (8.8) \\
\dot{\hat{\sigma}}(t) &= \Gamma \text{Proj}(\hat{\sigma}(t), -\hat{s}^\top(t) P b_m) \quad \hat{\sigma}(0) = \hat{\sigma}_0, \quad (8.9)
\end{align*}
\]

where \( \Gamma > 0 \) is an adaptation gain, \( \hat{s}(t) = \hat{s}(t) - s(t) \) is the prediction error, and \( \text{Proj}(\cdot, \cdot) \) is the projection operator defined in Definition B.3 in [59]. Symmetric positive definite matrix \( P \) is the solution of the algebraic Lyapunov equation \( A_m P + PA_m^\top = -Q \), given a symmetric positive definite \( Q \).
Adaptive control is designed using the adaptation states in (8.9) as follows:

$$\pi_{ad}(s) = -kD(s)(\hat{\eta}(s) - k_gr(s)),$$  \hspace{1cm} (8.10)

where $r(s)$ is the reference signal in the Laplacian form, and $D(s) = 1/s$ is a strictly proper transfer function that forms stable low-pass filter $F(s) = \frac{wkD(s)}{1+wkD(s)}$. The gain $k > 0$ is constant, and $k_g = -1/(c^\top A_m^{-1}b_m)$. The signal $\hat{\eta}(t)$ is obtained by $\hat{\eta}(t) = \hat{w}(t)\pi_{ad}(t) + \hat{\theta}^\top(t)s(t) + \hat{\sigma}(t)$.

**Design Control Parameters**

Now we design control parameters $\Gamma$, $k_m$, $P$, $V$, $k$, such that the proposed control input $\pi(t) = -k_m s(t) + \pi_{ad}(t)$ guarantees desired performance and robustness of the lateral state $x$ in (8.4).

We need to define the desired system behavior. Let us denote $s_{ref}$, $\pi_{ref}$ non-adaptive version control, i.e., the system behavior when (8.9) tracks the uncertainty perfectly. However, the control input cannot satisfy $w\pi_{ad}(t) + \theta^\top s(t) + \sigma(t) = 0$ because the perfect control input is filtered in (8.10) before the implementation. Let us denote $s_{des}$ and $\pi_{des}$ the design system having the ideal system behavior such that $w\pi_{ad}(t) + \theta^\top s(t) + \sigma(t) = 0$ holds for $\forall t$. Using the above definition, we can say that the system well-behaves if $\|s(t) - s_{des}(t)\|$ and $\|\pi(t) - \pi_{des}(t)\|$ are small enough.

We can choose an arbitrary large adaptation gain $\Gamma > 0$ so that the system performs arbitrarily close to the reference system ($s_{ref}(t)$ and $\pi_{ref}(t)$) by Theorem 2.2.2 in [59] without sacrificing robustness, where the reference system refers the $\mathcal{L}_1$ adaptive controller without adaptation. Then, the performance of the system is rendered as the error between the reference system and the design system ($\|s_{ref} - s_{des}\|_{\infty}$ and $\|\pi_{ref} - \pi_{des}\|_{\infty}$), where the design system is the ideal system that does not depend on the uncertainties. Since $A_m$ in (8.6) must be Hurwitz and $A_m(V)P + PA_m^\top(V) < 0$ should hold, we choose $k_m$ and $P$ such that $A_m(V)$ is Hurwitz and $A_m(V)P + PA_m^\top(V) < 0$ holds for all $V_{min} \leq V \leq V_{max}$, where $V_{max} \geq V_{min} \geq 0$ are the maximum and minimum velocity of the area.
Finally, we design \( V \) and \( k \) together balancing performance and robustness as follows:

\[
\max_{k, V \in [V_{\text{min}}, V_{\text{max}}]} \quad V
\]

\[
\text{s.t.} \quad \|G(s)\|_1 \leq \lambda_{gp}, \quad \forall w \in \Omega
\]

\[
k \leq \bar{k}
\]

(8.11)
(8.12)
(8.13)

for constants \( \bar{k} > 0 \), and \( \lambda_{gp} < \frac{1}{L} \), where \( G(s) = H(s)(1 - F(s)) \), \( H(s) = (sI - A_m)^{-1}b_m \) and \( L = \max_{\theta \in \Theta} \|\theta\|_1 \). The first constraint refers minimum performance guarantee and the second constraint indicates a minimum robustness guarantee, where \( r \) is the certified radius obtained by the classifier. By increasing \( k \), one can render \( \|G(s)\|_1 \) arbitrary close to zero and this improve the performance \( \|s_{\text{ref}} - s_{\text{des}}\|_\infty \) and \( \|\pi_{\text{ref}} - \pi_{\text{des}}\|_\infty \) (Lemma 2.1.4 in [59]). However, the time delay margin decreases as \( k \) increases. It is worth noting that the problem (8.13) is always feasible with \( V = 0 \).

The following result shows that the control algorithm we thus constructed (with the design parameters as chosen above) is robust in precisely the sense of Definition 8.4.2.

**Theorem 8.5.2.** (Robust Control Pipeline) Given a perturbed sensory input \( x + \delta \), if \( \delta \) is within a given certificate \( \tau \), the robust model \( g \) returns a robust prediction such that the corresponding cornering stiffness interval \( \zeta(g(x + \delta)) \) includes the true and nominal cornering stiffness. Assumption 8.3.1 holds. Therefore there exists positive constants \( c_1 \) and \( c_2 \) such that the constraints in definition 8.3.1 are satisfied, thus the end to end pipeline \( A(\zeta(g)) \) is robust per definition 8.4.2.

**Proof.** We defer the proof to the full version of the paper.

8.6 Experiments

In this section, we empirically study the robustness of the robust driving system described above with and without proposed formal end-to-end robustness certification across different weathers and road types, comparing the vulnerability of the non-robust driving system. We conduct experiments on three datasets, including driving frames from the Carla simulator [34] as well as the physical world (Road Traversing Knowledge (RTK) [117], robotCar [58]). These
datasets contain driving frames across four types of weather: sunny, light rain, heavy rain, and snow, and three different road surfaces: asphalt, cobblestone, and sand (in descending order of friction), with the range of road frictions from 30k to 120k. We use cornering stiffness to define road friction for lateral dynamic control. Typical cornering stiffness ranges from 20000 – 120000 N/rad, depending on many parameters such as road condition, rim size, and inflation pressure [44].

We consider two types of attacks by the attacker’s objective: (1) increasing the velocity, (2) decrease the velocity. We will refer these two types of attacks as Stability Attack (SA) and Efficiency Attack (EA). We consider velocity and instability as the car’s performance measurements. Specifically, velocity is the maximum safe velocity from $A$, and instability implies control system instability in Lyapunov sense. Intuitively speaking, a dynamic system is Lyapunov stable if it starts near an equilibrium point (center lane) and its trajectory stays near the equilibrium point forever. The higher the speed, the more efficient the car. The lower the instability, the more stable a vehicle is. Next, we separately discuss the Road condition Classification and Road Friction Regression problems. For each of the two problems, we start by showing the performance of the non-robust system $A(\zeta(f))$ in the unmodified environment, and we show the vulnerability of this non-robust system in malicious environments. Next, we show the efficacy of the certified robust system $A(\zeta(g))$ across malicious environments. We empirically show that this certified robust system $A(\zeta(g))$ ensures the car drives safely and efficiently in malicious driving environments.

<table>
<thead>
<tr>
<th></th>
<th>Carla</th>
<th>RTK</th>
<th>RobotCar</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accuracy</strong></td>
<td>98.6%</td>
<td>94.2%</td>
<td>95.6%</td>
</tr>
<tr>
<td><strong>Instability (SA)</strong></td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td><strong>Velocity (EA)</strong></td>
<td>29.42</td>
<td>28.45</td>
<td>28.46</td>
</tr>
</tbody>
</table>

Table 8.1: Accuracy and performance without malicious attacks

<table>
<thead>
<tr>
<th></th>
<th>Carla</th>
<th>RTK</th>
<th>RobotCar</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accuracy</strong></td>
<td>0%</td>
<td>80%</td>
<td>69%</td>
</tr>
<tr>
<td><strong>Instability (SA)</strong></td>
<td>200.00</td>
<td>37.50</td>
<td>61.50</td>
</tr>
<tr>
<td><strong>Velocity (EA)</strong></td>
<td>27.36</td>
<td>27.83</td>
<td>25.35</td>
</tr>
</tbody>
</table>

Table 8.2: Vulnerability of the non-robust perception model $f$.

### Road Condition Classification

The perception model takes the driving frame as input in the classification problem and predicts the weather or road types. Next, this predicted class is converted to a range of
cornering stiffness. Table 8.1 shows the accuracy of the non-robust perception model $f$ without malicious attacks. Table 8.2 shows the velocity and instability of the car driving in the unmodified environment, where the attacker doesn’t modify the environment. The first question we ask is *Is perception model $f$ vulnerable to malicious attacks?*, and to this, we answer *yes*. Without loss of generality, we use a common attack, *PGD attack* [95] as the malicious attacking approach. As a defender, we want the car to drive safely in a malicious environment. A model $g$ is robust if it only predicts classes with higher road friction than the victim class when presented with a corrupted image $x + \delta$. Table 8.2 shows the system $A(\zeta(f))$ is indeed vulnerable to malicious attacks. Now, we discuss the robustness of the robust system $A(\zeta(g))$. Table 8.3 empirically show the effectiveness of the robust perception model $g$ when defending against *Stability Attacks* and *Efficiency Attacks*, and provides the certificate of this robust model.

<table>
<thead>
<tr>
<th>Noise</th>
<th>Carla</th>
<th>RTK</th>
<th>RobotCar</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma$</td>
<td>Velocity</td>
<td>Certificate</td>
<td>Velocity</td>
</tr>
<tr>
<td>0.25</td>
<td>6.36</td>
<td>0.61</td>
<td>18.99</td>
</tr>
<tr>
<td>0.50</td>
<td>12.50</td>
<td>0.58</td>
<td>22.50</td>
</tr>
<tr>
<td>1.00</td>
<td>25.00</td>
<td>0.57</td>
<td>29.30</td>
</tr>
</tbody>
</table>

Table 8.3: Robustness: Instability (up) Efficiency (bottom) under attacks.

<table>
<thead>
<tr>
<th>Noise</th>
<th>Carla</th>
<th>RTK</th>
<th>RobotCar</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma$</td>
<td>Instability</td>
<td>Certificate</td>
<td>Instability</td>
</tr>
<tr>
<td>0.25</td>
<td>29.41</td>
<td>0.61</td>
<td>28.13</td>
</tr>
<tr>
<td>0.50</td>
<td>29.41</td>
<td>1.19</td>
<td>28.10</td>
</tr>
<tr>
<td>1.00</td>
<td>29.42</td>
<td>2.26</td>
<td>28.22</td>
</tr>
</tbody>
</table>

Table 8.4: Robustness: instability and efficiency

**Road Friction Regression**

Recall that each image in *Carla, RTK, Robotcar* datasets corresponds to a class. We convert each class to a corner stiffness.

We measure the vulnerability of $f$. Table 8.5 shows the mean square error and performance of $f$ without any attacks and under *PGD attack*. From these two tables, we observe $f$ is
Table 8.5: The MSE and performance of road friction regression in benign (up) and adversarial (bottom) environment.

Table 8.5: The MSE and performance of road friction regression in benign (up) and adversarial (bottom) environment.

malicious to adversarial attacks, as the MSE increases and performance decreases significantly. At last, table 8.4 shows combining the certified robust regression model $h$ with the robust control algorithm $A$ guarantees the stability and efficiency in the malicious environment.
Part IV

Conclusion
In this body of work, we have presented a series of publications that contribute to the field of autonomous driving, image recognition, and adversarial machine learning. Our research explores various aspects of security and robustness in autonomous systems, addressing challenges such as location spoofing attacks, certified robust control, adversarial examples in simulated driving, image provenance, and geolocation privacy protection.

The publication on location spoofing attacks highlights the vulnerabilities of autonomous fleets and emphasizes the need for robust security measures to mitigate the risks associated with adversarial attacks. Our work in certified robust control under adversarial perturbations establishes the importance of developing control strategies that can withstand intentional disruptions.

Furthermore, we propose innovative approaches for generating adversarial examples in simulated autonomous driving scenarios, shedding light on the vulnerabilities of image recognition models and the potential risks they pose in real-world applications. Our work on image provenance and geolocation privacy protection addresses the privacy concerns associated with photo collections and emphasizes the significance of preserving individuals’ privacy in the digital age.

These publications contribute to the broader field of autonomous driving and image recognition by uncovering potential security threats and proposing strategies to enhance the robustness and privacy of these systems. We believe that our research provides valuable insights and lays the foundation for future advancements in securing autonomous systems and advancing the field of adversarial machine learning.

In conclusion, our publications collectively address critical challenges in autonomous driving and image recognition, offering novel solutions and insights. We anticipate that our work will inspire further research and foster the development of more secure, reliable, and privacy-preserving autonomous systems in the future.

### 8.7 Publications

1. Location Spoofing Attacks on Autonomous Fleets. Jinghan Yang, Andrew Estornell, Yevgeniy Vorobeychik, *NDSS VehicleSec 2023*


4. "PROVES: Establishing Image Provenance using Semantic Signatures" - Authors: Mingyang Xie, Manav Kulshrestha, Shaojie Wang, Jinghan Yang, Ayan Chakrabarti, Ning Zhang, Yevgeniy Vorobeychik *WACV 2021*

5. Protecting Geolocation Privacy of Photo Collections, Jinghan Yang, Ayan Chakrabarti, Yevgeniy Vorobeychik, *AAAI 2020*

Part V

References
References

[1] Waymo has launched its commercial self-driving service in phoenix — and it’s called ‘waymo one’.


[70] K. Korosec. Waymo’s driverless taxi service can now be accessed on google maps, 2021.


