Runtime Stealthy Perception Attacks against DNN-based Adaptive Cruise Control Systems

Xugui Zhou
xuguizhou@lsu.edu
Louisiana State University, USA

Anqi Chen
chen.anqi3@northeastern.edu
Northeastern University, USA

Maxfield Kouzel
Haotian Ren
University of Virginia, USA

Morgan McCarty
morgannmccarty@gmail.com
Northeastern University, USA

Cristina Nita-Rotaru
c.nitarotaru@northeastern.edu
Northeastern University, USA

Homa Alemzadeh
ha4d@virginia.edu
University of Virginia, USA

Abstract

Adaptive Cruise Control (ACC) is a widely used driver assistance technology for maintaining desired speed and safe distance to the leading vehicle. This paper evaluates the security of the deep neural network (DNN) based ACC systems under runtime stealthy perception attacks that strategically inject perturbations into camera data to cause forward collisions. We present a context-aware strategy for the selection of the most critical times for triggering the attacks and a novel optimization-based method for the adaptive generation of image perturbations at runtime. We evaluate the effectiveness of the proposed attack using an actual vehicle, a publicly available driving dataset, and a realistic simulation platform with the control software from a production ACC system, a physical-world driving simulator, and interventions by the human driver and safety features such as Advanced Emergency Braking System (AEBS). Experimental results show that the proposed attack achieves 142.9 times higher success rate in causing hazards and 82.6% higher evasion rate than baselines, while being stealthy and robust to real-world factors and dynamic changes in the environment. This study highlights the role of human drivers and basic safety mechanisms in preventing attacks.

CCS Concepts

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Security and privacy** → **Systems security**.

Keywords

Runtime Attack, Safety Intervention, AEBS, ADAS, ACC, DNN, Perception Attack, Stealthy.

ACM Reference Format:

Xugui Zhou, Anqi Chen, Maxfield Kouzel, Haotian Ren, Morgan McCarty, Cristina Nita-Rotaru, and Homa Alemzadeh. 2025. Runtime Stealthy Perception Attacks against DNN-based Adaptive Cruise Control Systems . In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '25), August 25–29, 2025, Hanoi, Vietnam.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3708821.3710832

ASIA CCS '25, August 25–29, 2025, Hanoi, Vietnam

ACM ISBN 979-8-4007-1410-8/25/08
https://doi.org/10.1145/3708821.3710832

1 Introduction

Level-2 Advanced Driver Assistance Systems (ADAS) provide autonomous driving features while still requiring human attention at all times [1]. Examples include Adaptive Cruise Control (ACC) which controls longitudinal movement, Automatic Lane Centering (ALC) which controls lateral movement, and Advanced Emergency Braking System (AEBS) which controls braking through Automatic Emergency Braking (AEB) and provides warnings through Forward Collision Warning (FCW). Over 17 million passenger cars worldwide are equipped with ADAS [2].

One important ADAS feature is ACC, which makes highway driving more comfortable by automatically changing the speed when traffic slows down or speeds up. ACC takes as input sensor measurements such as radar, Lidar, or camera and adjusts the speed to maintain a safe following distance to the lead vehicle [3, 4]. At the core of ACC lies the detection and tracking of the lead vehicle. Highly accurate methods [5, 6] for detection and tracking rely on Deep Learning (DL) based object detection using camera or fusion of camera and radar/Lidar data. A Longitudinal planner (LP) uses the prediction from the DL module to compute the desired speed and acceleration. Malfunctioning of the object detection module can have serious consequences including accidents. Given the critical role of object detection and tracking in the safety of ACC and that many commercial ACC systems (e.g., Tesla Autopilot [7], Comma.ai OpenPilot [8]) use DL-based object detection, these mechanisms must function correctly under any conditions, including in the presence of adversaries.

Previous work has shown attacks against Deep Neural Networks (DNN) used for perception such as adversarial perturbations [9], adversarial patches [10–12] or well-crafted stickers on road signs [13], the road [14], or camera lenses [15] to change the predicted class or probability of detecting a target object or the lane lines. However, misprediction of the lane lines only affects the lateral control (ALC) and simply changing the lead object class or detection probability does not necessarily impact the LP enough to cause unsafe ACC behavior (e.g., sudden acceleration). Attacking DNN-based ACC systems necessitates influencing the relative distance and speed with respect to the lead vehicle. One such attack using physical adversarial patches to control the relative distance and speed to the lead vehicle was shown against a production ACC [16]. However, the attack required placing a conspicuous large patch on the back of a truck and driving the truck in front of the target vehicle, a method easily noticeable or preventable by human drivers (see Appendix D).

In terms of computational effort, the perturbation-based attacks described above predominantly rely on *offline* optimizations. For attacks on ACC to be effective and robust, perturbations must be created at *runtime* by considering dynamic factors such as the lead vehicle's size and position across consecutive camera frames, interdependencies across the frames, and environmental changes, while satisfying runtime computation constraints.

Notably, previous work ignored the presence of safety interventions in the ADAS control loop. Although some attack works have considered AEB or FCW, they either do not apply to ACC [12] or directly change the state estimations [17]. Some recent works [18, 19] have used the system context to determine the optimum time for attack injection. However, they did not directly compute the perturbations for the DL perception module to affect relative speed and distance and could be easily detected by existing safety mechanisms or anomaly detection methods [20, 21].

To fill these gaps, in this paper, we focus on runtime stealthy attacks against DNN-based ACC systems that inject minimal image perturbations into the DNN input with the goal of causing ACC controller to issue unsafe acceleration commands that cannot be mitigated by the human driver or existing safety mechanisms and lead to safety hazards, such as forward collision. We assume baseline ACC, ADAS software, and existing safety mechanisms are trusted.

Designing *stealthy safety-critical attacks* in the *human-in-the-loop ADAS* is a challenging task as the attackers need to explore the extensive attack parameter space to devise a strategy for effectively manipulating the DNN inputs and causing unsafe driving behavior while considering the dynamic changes of the environment at runtime, real-time constraints, and safety interventions. We propose a *context-aware* attack strategy and *optimization* method together with a *safety intervention* simulator to explore key questions on how the *timing* and *value* of perturbations affect the success of attacks in (1) causing safety hazards and (2) evading human driver or safety mechanism detection and intervention. To the best of our knowledge, this is the first evaluation of the security of DNN-based production ACC systems under *runtime strategic attacks* using a *combined knowledge-and-data-driven* approach by taking into account *human drivers* and *realistic safety mechanisms*. This study provides insights into the vulnerabilities and risks associated with DNN-based ACC systems and the role of human operators and safety mechanisms in preventing attacks.

The main contributions of the paper are the following:

- We adopt a control-theoretic hazard analysis method to identify the most critical system contexts for launching attacks that maximize the chance of forward collisions.
- We design a novel optimization-based approach and an adaptive algorithm to generate stealthy image perturbations and add them in the form of an adversarial patch to the input camera frames at *runtime* to fool DNN model and cause unsafe acceleration by ACC controller before being detected or mitigated by the human driver or ADAS safety mechanisms.
- We evaluate the effectiveness of the attacks with a real vehicle and driving dataset and a realistic simulation platform that integrates an open-source ADAS control software, OpenPilot from Comma.ai (with over 10,000 active users on the road) [8] and the state-of-the-art physical-world driving simulator, CARLA, with a driver reaction simulator and the typical ADAS safety mechanisms (AEB and FCW), which we implement based on the international standards.
- Experimental results show that our context-aware attack strategy causes 28.6x more hazards than random attacks. The proposed optimization-based perturbation algorithm achieves a 100% attack success rate in four high-risk driving scenarios (in the absence of safety interventions), 142.9x and 1.9x higher than random value perturbation and APGD-based methods [22]. Our approach is also stealthy and robust to real-world factors such as different camera positions, distances to lead vehicle, and weather and lighting conditions.
- We observe similar results in the presence of the human driver and safety feature interventions, where our attack still achieves an 82.6% success rate, while all the random and APGD-based attacks are mitigated by the safety mechanisms.

Ethics. We have submitted responsible disclosures to Comma.ai. For the human subject study, we received IRB approval and followed the IRB requirements for the recruitment of participants and conduct of experiments.

2 ADAS Overview

Fig. 1 shows the overall structure of a typical ADAS, including ACC, AEBS, and ALC features.

2.1 Adaptive Cruise Control (ACC)

The main goal of ACC is to maintain a safe following distance between the autonomous vehicle (referred to as Ego vehicle or AV) and the vehicle driving in the same lane in front of the AV (referred to as lead vehicle or LV) by adjusting the AV speed based on the estimated relative distance and relative speed to the LV.

Sensors. Existing DNN-based ACC systems either use camera data (e.g., Tesla Autopilot, Subaru Eyesight) or both camera and radar data (e.g., Apollo [23] and OpenPilot) to predict and track LVs and objects. Other sensors, such as GPS or IMU, are also used to detect current speed to match the target speed set by the drivers.

Lead Vehicle Detection. The most critical part of ACC is lead vehicle detection (LVD), which estimates the relative speed (*RS*) and distance (*RD*) to the LV using camera data or a fusion of camera and radar data. Sensor fusion is the process of combining measurements from multiple sensors (e.g., camera and radar) usually using a Kalman filter [24] to overcome the limitations of individual sensors and obtain a more accurate perception of the surrounding environment. Based on the LVD outputs, the main driving control actions (i.e., acceleration, deceleration, braking) are determined.

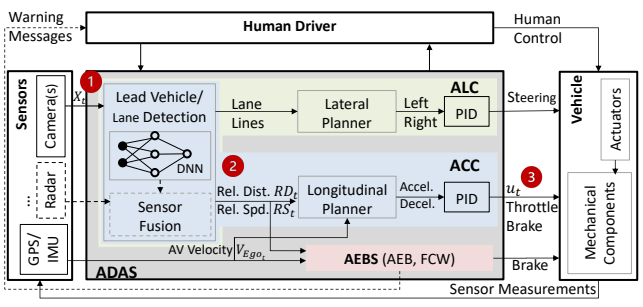

Figure 1: ADAS architecture with ACC, AEBS, and ALC.

Table 1: Comparison of attacks on DNN-based ADAS.

Attack	Attack			Safety Interventions		Autonomy
Method	Type	Vector	Target	AEBS	Driver	Level[1]
[13]	Offline	Stickers on road signs	Classifier	N	N	N/A
[27]		Patch projected	MTO	N	N	L4
[14]		Stickers on road	ALC	N	N	L2
[16]		Patch on truck	ACC	N	N	L2
[12]		Patch on road	ALC	**Y**	N	L2
[28]	Runtime	Perception inputs	MTO	N	N	L4
[19]		Perception inputs	MTO	No FCW	N	L4
[17]		Inner state variables	FCW	No AEB	Y	N/A
[18]		Control commands	ACC, ALC	No AEB	**Y**	L2
Ours		Perception inputs	ACC	**Y**	**Y**	L2

* MTO: Multi-Object Tracking;

Longitudinal Planner. The next stage involves determining the optimal speed based on the LVD outputs and current vehicle state. The longitudinal planner uses algorithms such as Model Predictive Control (MPC) to generate multiple desired speed trajectories, each representing a series of speeds in a certain following period [25].

Vehicle Control. At each control cycle t, the plan from the longitudinal planner with the lowest speed and risk (e.g., risk of colliding with the LV) is selected by the ACC system and fed to a Propotional-Integral-Derivative (PID) [26] controller to get the specific optimal control command u_t in the form of the throttle or brake amount such that the vehicle accurately and quickly follows the desired speed trajectory. Upon execution of the control command by the actuators, the vehicle's physical state, s_t (e.g., current speed, location), transitions to a new state s_{t+1}.

2.2 ADAS Safety Mechanisms

The Advanced Emergency Braking System (AEBS), including FCW and AEB, is a fundamental ADAS safety mechanism that alerts drivers about potential collision risks with a lead obstacle and actively decelerates the vehicle to prevent accidents. As shown in Fig. 1, most AEBS implementations utilize both camera and radar for collision prediction through sensor fusion [29] and make control actions based on the LVD outputs and other sensor measurements (see Appendix C). In addition, some safety principles (such as maximum acceleration limits as required by international standards, e.g., ISO 22179) or firmware safety checks (e.g., constraints on the output steering angle) are incorporated into the design of typical production ADAS to ensure driving safety [30].

Previous studies on security of autonmous driving either focus on Level 4 or fully autonomous vehicles without considering the impact of the human driver interventions during an emergency situation (e.g., abnormal acceleration) or have overlooked the inclusion of basic safety features like AEB or FCW and their impact on the attack effectiveness (see Table 1). For a realistic assessment of ACC security, it is essential to evaluate the interventions of these safety features. At Level 2, drivers must maintain control and supervise ADAS functionalities [1]. There exists a research gap concerning how to make the combination of a human driver and an autonomous vehicle acceptably safe. The primary challenge lies in assessing the ability of human drivers to anticipate and respond to situations where automation may fail.

2.3 OpenPilot

We use a production ADAS called OpenPilot from Comma.ai [8] as our case study. OpenPilot is the only open-source production Level-2 ADAS, designed with the goal of improving visual perception and automated control (with ACC and ALC) through installing custom hardware to the OBD-II port on a vehicle. The targeted ACC system in OpenPilot follows the typical DNN-based ACC system architecture described in Fig. 1 with an end-to-end system design [31]. Currently, OpenPilot supports over 250 car models (e.g., Toyota, Honda, etc.) [32], has more than 10,000 active users and has accumulated a total driving distance of over 100 million miles on actual roads [8]. It is reported to achieve state-of-the-art autonomous driving performance, beating 17 existing production ADAS on the market in overall ranking by Consumer Reports [33].

The DNN model used by OpenPilot, called Supercombo, utilizes an EfficientNet-B2 based CNN model to process image data [34]. It incorporates the state of the vehicle and the environment by adding additional inputs from traffic conventions and the desired state. Multiple branches of GEMM (General Matrix Multiply) operations are then used to derive various predictions, such as lane lines, LVs, and vehicle pose, resulting in a total of 6,472 outputs [35].

Comma.ai offers the community a closed-loop simulation environment [8] that integrates OpenPilot with a physical-world open urban driving simulator called CARLA, which can generate near-real high-quality camera image frames of the environment and has been widely used in the literature of autonomous driving [36]. However, in this default simulation, the sensor fusion only relies on the camera data since no radar sensor is available. Further, none of the typical ADAS safety mechanisms are included.

3 Runtime Context-Aware Perception Attack

In this section, we introduce our attack model, attack challenges, and runtime stealthy perception attack design.

3.1 Attack Model

Attacker Objective. The objective of the attacker is to maximize the error in LV predictions by the LVD's DNN module and cause forward collisions, while remaining stealthy to avoid being detected or prevented by driver or safety mechanisms (e.g., AEBS).

The attack is crafted in a stealthy way such that it is not distinguishable from noise, human errors, or accidents. This enables it to remain hidden longer and makes it less easily detected/prevented by existing defense mechanisms (e.g., anomaly detection [37] or input transformation [38–40, 40]). The attacker can accomplish this by targeting the DNN inputs (1 in Fig. 1) or directly manipulating the DNN outputs (2), depending on their capabilities and access level to the ACC system. In this paper, we primarily focus on DNN inputs to enhance stealthiness, as detailed in Section 5.4.

Attacker Knowledge. We assume the attacker gains comprehensive knowledge about the target ACC system design and implementation by reverse engineering a purchased or rented vehicle with identical control software as the victim vehicle [14, 37] or by studying publicly available documents or source code. This is possible given some production ACC systems are open source [8, 23].

Attacker Capabilities. We assume the attacker has the capability to *intercept and change live camera image frames* at runtime to compute an adversarial patch and fool the DNN model of ACC.

A possible way to achieve this is to implant malware by compromising the over-the-air (OTA) update mechanisms [41–44] or gaining one-time remote access to the ADAS software through scanning

Table 2: Threat models: attacker strength, capability, and impact.

Threat Model	Attacker Strength	Access to ADAS Software	Vehicular Networks	Computation Location	Impact	Examples
Malware	Strong[1]	✓	r/w*	within ADAS	Fleet of Vehicles	[44, 52]
Wireless	Medium[2]		r/w	Local Device, Remote Server	Single Vehicle	[53][54][19] [46][55]
Physical	Weak[3]		r	Remote Server	Single Vehicle	[56][57] [58][59][60]

[1] Other malware attacks are possible (e.g., DNN output, controller output);
[2] Other sensor/actuator attacks are possible (e.g., RADAR, GPS, controller output);
[3] Only perception attacks possible;
* r/w represents read (r) and write (w) access to vehicular networks.

the network, accessing stolen credentials and exploiting the vulnerabilities in SSH protocol [45], browsers, access control [46], wireless communications [46–48], third-party components connected to in-vehicular network [49], or some remote service/backdoor offered by the manufacturer (e.g., Comma Connect for OpenPilot [50] or Bluelink for Hyundai). For example, a publicly-available tool developed for OpenPilot enables an attacker on the same network as a target device to install a malicious code [51]. With such remote access, the attacker can also change the OTA settings (e.g., remote URL) to prevent potential patches from being effective. This assumption about the attack surface for deploying malware is also supported by previous works [44, 52], and could have a large impact as it can be generalized to any vehicle with similar OTA and DNN mechanisms and target a large fleet of vehicles at the same time.

Another way to compromise live camera data is to connect to a wireless communication device, either a third-party component or one implemented by an attacker, connected to the vehicular network, such as ROS communication channels [53], CAN Bus [42, 46, 55, 61] or Ethernet channel [19, 54]), to read and send image data at runtime. The attacker computes the attack value on a local wireless device or a remote server.

Further, physical attack methods are also viable, such as by displaying the patch on a monitor attached on the rear side of a leading adversarial vehicle [56, 57] or projecting the patch into the rear of the LV using a projector [58–60].

Table 2 summarizes various methods for runtime reading and modifying of live camera frames, given different attacker strengths and capabilities. In this paper, we mainly focus on the runtime and optimized modification of live camera frames to enhance attack success rate and stealthiness, regardless of the threat model and how the attacker obtained access. In our experiments, we implement the attack through malicious OTA update to OpenPilot (Section 3.3).

Attacker Constraints. We restrict the scope of attack capabilities in the paper to reading and modifying live camera data, ensuring uniformity across all presented threat models. Although some attack models could potentially enable more aggressive attacks, such as directly altering ACC controller outputs (3 in Fig. 1) via malware or wireless method or changing the DNN output (2) with malware method, these may be easier or earlier identified by safety mechanisms (e.g., AEBS) or human driver (see Section 5.4).

The attacker does not consider injecting or replaying pre-recorded fake video frames as they need to be perfectly engineered offline or be pre-recorded, would not suit the constantly changing environment (e.g., surrounding vehicles, road conditions) at runtime, and could be easily noticed by human drivers (see Appendix E).

3.2 Attack Challenges

Several challenges need to be addressed in attacking DNN-based ACC systems at runtime.

C1. Optimal timing of attacks at runtime to cause safety hazards. Prior attacks on ADAS that rely on random strategies to determine the attack timing (start time and duration) have proven ineffective in achieving a high attack success rate [18, 62, 63] as they waste computational resources by trying random attack parameters that lead to no safety hazards. For instance, initiating an attack on an Ego vehicle to induce sudden acceleration does not cause safety hazards when no LV is detected. Recent works have focused on using machine learning to explore the fault/attack parameter space [64] and improve the attack effectivenes [19, 63], but they still require substantial amounts of data from random attack experiments for model training. Finding the optimal triggering time and duration is crucial for effective attacks, yet challenging due to the vast parameter space that needs exploration.

C2. Generating attack value at runtime to adapt to dynamic changes in the driving environment. Attacking DNN-based ACC systems on a moving vehicle faces challenges due to continuous variations in the driving environment, such as object position and size captured by the Ego vehicle's camera. Existing attack algorithms [13, 16] are inadequate as they plan perturbations offline, assuming fixed sizes and locations for attack vectors. A new algorithm is needed to dynamically adapt the attack vector's value (e.g., position, dimension, and amount of perturbation) to match the LV's dynamics. These changes disrupt the original attack vector generation process, requiring a unique approach to address inconsistencies and non-differentiability in the objective function. In addition, the attack value should be designed in a stealthy way to avoid detection by the human driver or safety mechanisms.

C3. Incorporating real-time constraints into the attack optimization process. Previous attacks on DNN models assume predetermined target images [13] or a known set [12, 16] with unlimited computation resources, allowing iterative optimization until an optimal attack vector is generated. However, attacking ACC systems in real-time presents challenges as the camera continuously provides frames without prior knowledge. An attack vector must be generated in real-time before the next frame or control action execution. The real-time control cycle and camera update frequency limit the speed of generating the adversarial attack vector and the frequency of assessing the perturbation's impact on DNN predictions. These tight constraints in typical ACC systems (e.g., frame rate of 20Hz and control cycle of 10ms in [8]) significantly impact the effectiveness of optimization-based attack strategies.

3.3 Attack Design

Fig. 2 illustrates the overall design of our attack, including the steps during the runtime execution and offline preparation.

To tackle challenge **C1**, rather than randomly or exhaustively exploring the attack parameter space, we systematically characterize specific values within the parameter space (e.g., attack start time and duration). This targeted approach aims to identify optimal *system contexts* (or critical times) for activating the attacks to not waste time and resources on non-hazardous scenarios.

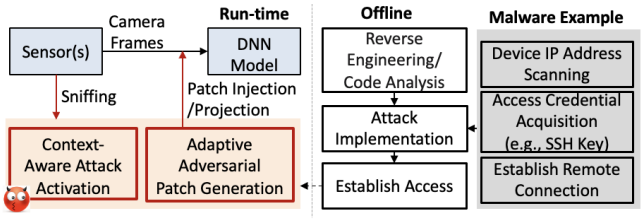

Figure 2: Attack design: Offline preparation, Runtime execution.

To address the challenges **C2-C3**, we design a novel optimization approach and an adaptive algorithm to dynamically determine optimal pixel values for an adversarial patch at runtime, aiming to maximize the error in DNN-based LV predictions and to accommodate dynamic changes in the driving environment. To optimize attack effectiveness and computational efficiency, our method focuses on the small area of the bounding box around the target vehicle and employs a primary attribution algorithm [65] to identify and manipulate the most crucial pixels (Section 3.3.2). Additionally, our adaptive algorithm retains optimization results of perturbation size, position, and value from the previous perception cycle instead of restarting the optimization process (contrary to other iterative optimization methods [22]) to satisfy the tight real-time constraints of the perception system (50ms).

The attacker initiates the attack process by conducting an offline analysis on the target DNN-based ACC system. The analysis includes examining operational data and open-source code to identify the target software files and functions and/or the DNN input and output fields to be monitored and infected. Then the attack steps outlined in the orange box in Fig. 2, are implemented and executed either remotely or locally through unauthorized access to the target ACC under one of the threat models described in Table 2. In Fig 2, we present an attack implementation example via malware method that will replace the target functions or system libraries. The attacker installs the malware on the target ACC system through one-time access to the victim vehicle control system, achieved by exploiting remote access vulnerabilities or compromising OTA updates (Section 3.1). Appendix B provides an illustrative example of how we install the malware on an OpenPilot system. At runtime, the malicious code intercepts sensor data before reaching the DNN model and infers the current system context for activating attacks. When the detected system context aligns with a critical system context (Section 3.3.1), an optimized adversarial patch is generated (Section 3.3.2) and added to the image data and sent to DNN model.

3.3.1 Context-Aware Attack Activation. To find the most critical times for activation of the attack (**C1**), we adopt a control-theoretic hazard analysis method [66] to identify the most critical system contexts under which specific control actions are unsafe and, if issued by the ACC control software, could lead to hazards. This approach mainly relies on domain knowledge about system safety requirements and, contrary to an ML-based approach, does not require large amounts of training data or computation resources.

In our hazard analysis, we define the accident as the adverse event of forward collision (measured by a zero or negative relative distance between the Ego vehicle and the LV or a front object). This can happen as a result of the Ego vehicle transitioning into a hazardous state that violates the safe following distance with LV.

Table 3: Partial safety context table for an ACC system.

Rule	System Context		Control Action	Potential Hazards?
1	HWT⩽HWT$_{safe}$	RS⩽0	Acceleration	No
2		RS>0		Yes
3	HWT>HWT$_{safe}$	RS⩽0		No
4		RS>0		No

* HWT: Headway Time = Relative Distance/Current Speed;
* RS: Relative Speed = Current Speed (V_{Ego}) - Lead Speed (V_{Lead});

To determine the critical system contexts, we assess all the combinations of system states and ACC control actions (e.g., acceleration, deceleration) to identify the specific combinations that are most likely to lead to hazards. Table 3 shows part of the safety context table for ACC with a focus on the acceleration commands. Here, the critical system context is described as when the Headway Time (HWT, the time the Ego vehicle takes to drive the relative distance (RD) from the LV with the current speed) is less than a safety limit HWT$_{safe}$ (e.g., 2-3s), and the Ego vehicle is faster than the LV ($V_{Ego} > V_{Lead}$). Under such system context, an acceleration command induced by the camera perception attacks or other reasons will most likely lead to a forward collision hazard. This high-level specification of critical system context can be done by an attacker based on the knowledge of the typical functionality of an ACC system and be applied to any ACC system with the same functional specification.

3.3.2 Adaptive Adversarial Patch Generation. The critical system contexts identified in the previous section are based on the high-level unsafe actions (e.g., Acceleration) issued by the ACC controller. In order to find the specific attack values or DNN input perturbations that can cause such unsafe control actions, we present an optimization-based patch generation method as shown in Fig. 3.

Runtime Optimization-based Adversarial Patch Generation. To address challenge **C2**, we formulate the attack as the following runtime optimization problem:

$$\min \sum_{d \in RD_t} -\nabla g(d,\theta) + \lambda ||\Delta_t||_p \tag{1}$$

$$s.t.\ Patch_t = \Delta_t * M_t \tag{2}$$

$$Patch_t \in [\mu - \sigma, \mu + \sigma] \tag{3}$$

$$Area(Patch_t) \subset BBox(LV)_t \tag{4}$$

$$X_t^{adv} = X_t + Patch_t \tag{5}$$

$$[RD, RS]_t = LVD_\theta(X_{t-1}^{adv}) \tag{6}$$

$$u_t = ACC(s_t, [RD, RS]_t) \tag{7}$$

$$s_{t+1} = CarModel(s_t, u_t) \tag{8}$$

where Eq. 1 is an objective function that aims at accelerating the Ego vehicle as soon as possible to cause a forward collision.Directly decreasing the probability of a lead vehicle or its bounding box (BBox) cannot change the ACC system behavior. We instead design an objective function that increases RD_t as much as possible while keeping the perturbation value of the adversarial patch imperceptible to human eyes.

In Eq. 1, $g(d)$ is an approximate polynomial function of d that fits the trend of the trajectory of the relative distance RD_t, predicted by the DNN model with weight parameters θ. "-" is a negative sign that

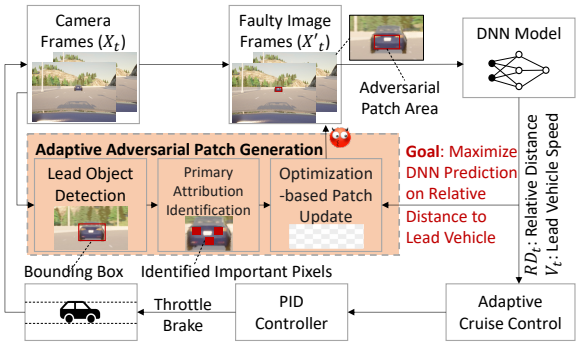

Figure 3: Optimization-based adversarial patch generation.

converts our goal of maximizing the relative distance to minimizing the proposed objective function. For example, when the gradient of the relative distance trajectory, $g(d)$, is negative, minimizing "$-\nabla g(d)$" will slow down the decrease of $g(d)$ and is equivalent to maximizing the relative distance. We adopt the gradient of $g(d)$ in the objective function instead of using RD_t itself in order to avoid sharp changes in the predicted relative distance value, which might be easily detected by some anomaly detection mechanisms. We assume the attacker has access to the DNN predictions (e.g., RD) by monitoring the ADAS communication network (e.g., ROS) or by running a replicated DNN model on a remote server or wireless communication device (see Table 2).

In Eq. 5, the perturbation is added to the original image input $X_t \in \mathbb{R}^{H\times W\times C}$ in the form of an adversarial patch $Patch_t \in \mathbb{R}^{H\times W\times C}$, represented as a matrix of pixels with height H, width W, and C color channels. λ is the weight parameter of the p-norm regularization term, designed to minimize the perturbation value of the patch for stealthiness. We limit the perturbation value within the Kalman filter noise parameters (μ,σ) (Eq. 3), which ensures the perturbation is not corrected by the sensor fusion. We also constrain the adversarial patch inside the BBox of the LV (Eq. 4) to enhance attack effectiveness, minimize the perturbation area for stealthiness, and reduce computational cost.

Primary Attribution Detection and Patch Update. As mentioned in Section 3.2, a major challenge (**C3**) in the design of runtime attacks is the changes in the size and location of the LV in the perceived image frames. To address this challenge, the attacker needs to update the generated patch dynamically according to the approximate DNN outputs. In this paper, we adopt an object detection method [67] to detect and track the real-time location and size of the LV and concentrate the attack perturbation within the detected BBox of the LV. In production ACC with object detection features [23], given proper access, the attacker can skip this step and use the stock prediction results.

After getting the BBox, we utilize a primary attribution algorithm [65] to quantify the relationship between input features and output predictions. Through this exploration, we try to identify the important pixels within the BBox of the LV that contribute the most to the predictions of RD_t. The input pixels with high weights identified by the attribution algorithm are marked by unit value in the mask matrix M_t, and the remaining pixels are assigned zero values. This mask matrix is then multiplied by the perturbation Δ_t to generate the adversarial $Patch_t$ (in Eq. 2). This step is useful as it can filter out non-important pixels in the inputs to reduce the number of perturbed pixels to improve the effectiveness of optimization-based attacks and reduce computation costs of runtime attacks.

Finally, we develop a new initialization algorithm to shift the patch position and adjust its size when the detected BBox changes (Eq. 9-11). We shift the attack vector toward the new position of the detected BBox of the LV with a magnitude of $(x_t - x_{t-1}, y_t - y_{t-1})$, where (x_{t-1}, y_{t-1}) and (x_t, y_t) are the centers of BBox at previous and current control cycles, respectively (Eq. 9). We then expand the adversarial patch attack vector ($Patch$) to the dimension that matches the size of the newly detected BBox of the LV. Instead of reinitializing the whole attack vector matrix with random or zero values, which will reset the whole optimization process, we keep the previous patch values and intermediate variables and only initialize newly expended units (Eq. 10-11). Fig. 4 shows an example.

$$Pos(Patch_t) = Pos(Patch_{t-1}) + (x_t - x_{t-1}, y_t - y_{t-1}) \tag{9}$$

$$Init(\Delta_t) = [0] * size(BBox(LV)_t) + \begin{bmatrix} \Delta_{t-1} & 0 \\ 0 & 0 \end{bmatrix} \tag{10}$$

$$Init(Patch_t) = Init(\Delta_t) * M \tag{11}$$

This algorithm maintains a continuous optimization process across two consecutive perception cycles, which is critical in satisfying the real-time constraints. Fig. 5 shows a visualization of how the adversarial patch affects the DNN predictions.

4 Safety Intervention Simulation

To evaluate the safety of DNN-based ACC systems under attacks, we enhance the default OpenPilot and CARLA simulation platform (see Section 2.3) to be more representative of real-world ADAS, by developing a safety intervention simulator and mechanisms for priority-based dispatching of control commands to CARLA and fusion of camera and radar data (see Appendix A). An overview of the simulation platform is shown in Fig. 6 (with the orange parts representing our new implementations) and presented next.

To fill the gap in considering safety interventions and address the challenge of ensuring the combination of human driver and vehicle safe (see Section 2.2), we implement and integrate three levels of safety interventions in the OpenPilot software (see Fig. 6), including ADAS safety features (e.g., AEB and FCW), basic car safety constraint checking on control commands, and driver interventions.

AEBS (FCW and AEB) Simulator. To design and test the AEBS mechanisms in simulation, we thoroughly review the regulations and requirements concerning AEBS [68] [69] [70] and adhere to UN Regulation No. 152 [69]. We adopt and implement a time-to-collision (TTC) based phase-controlled AEBS [71] in our platform.

The AEBS processes inputs from LVD outputs (after sensor fusion), including relative distance (RD) and relative speed (RS), and current speed V_{Ego} (see Fig. 1). The average driver reaction time (T_{react}) is set to the commonly used constant value of 2.5s [12, 18]. Time thresholds, namely ttc (time to collision), t_{fcw} (FCW time), t_{pb1} (1st phase partial brake time), t_{pb2} (second phase partial brake time), and t_{fb} (full brake time), are then calculated as follows:

$$ttc = RD/RS; \tag{12}$$

$$t_{fcw} = T_{react} + V_{Ego}/4.5 \tag{13}$$

$$t_{pb1} = V_{Ego}/2.8; t_{pb2} = V_{Ego}/5.8; t_{fb} = V_{Ego}/9.8 \tag{14}$$

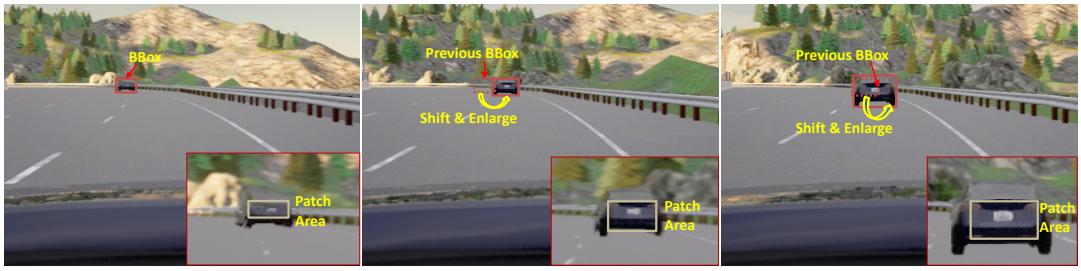

Figure 4: Examples of the shift and adjustment process in the patch generation. Inset figures are the zoomed-in views of the front vehicle with an adversarial patch added around the license plate area.

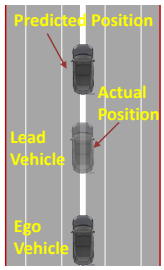

Figure 5: ACC under attack.

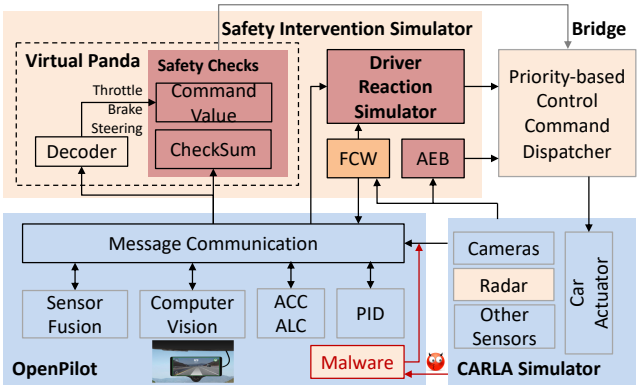

Figure 6: Simulation platform. [Code Available: https://github.com/gitguige/openpilot0.8.9]

Table 4: Driver simulator: activation conditions and reactions.

Activation Condition	Driver Reaction	Reaction Time
Alerts (e.g., FCW) Unexpected Acceleration Unsafe Cruise Speed Unsafe Following Distance Obvious Camera Perturbation	Emergency Brake & Zero Throttle No changes in the steering angle	2.5 seconds
Hard Braking	Stop brake and output regular throttle No changes in the steering angle	2.5 seconds

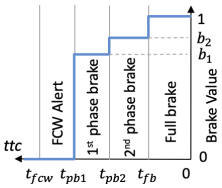

Figure 7: AEBS.

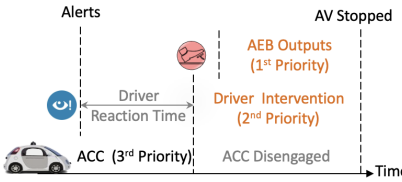

Figure 8: Control command dispatcher.

As shown int Fig. 7, when ttc falls below t_{fcw}, t_{pb1}, t_{pb1}, and t_{pb1}, a corresponding action (warning or brake with 90%, 95%, 100% force) is executed. Applying the brake blocks other ADAS controls. See Appendix C for more details on AEBS design and testing.

In reality, when OpenPilot is installed on a car, some car models lose the AEBS functionality [32], while others retain it. Also, AEBS might rely on a separate ADAS camera [72], distinct from the OpenPilot camera, and a potential compromise of the AEBS camera data is possible. Thus, we consider three scenarios for AEBS interventions: (1) AEBS is enabled, and AEBS camera data is uncompromised; (2) AEBS is enabled, but AEBS camera data is compromised; and (3) AEBS is disabled (see Section 5.3.2 and Table 7).

Safety Constraint Checker. The OpenPilot safety mechanisms are implemented in its control software and the Panda CAN interface. Panda is a universal OBD adapter developed by Comma.ai [73] that provides access to almost all car sensors through the CAN bus and also enforces some safety constraints over output commands. However, when integrated with the CARLA driving simulator, OpenPilot does not utilize Panda software or hardware; thus Panda safety checks are inactive.

To be as realistic as the actual OpenPilot on the road, we add a *virtual Panda* module in our simulation that copies the exact logic of Panda software [73]. Specifically, as shown in Fig. 6, the virtual Panda decodes the CAN packages sent by the ACC and checks whether their checksum is correct and the control command values are within predefined thresholds [73]. For example, to ensure safety, the maximum acceleration and deceleration of the vehicle shall be limited between $2m/s^2$ and -$3.5m/s^2$ respectively [30]. Only the commands that pass the Panda safety checks are sent to the simulated vehicle actuators. In CARLA simulator, the final control commands are truncated within the range of [0,1].

Driver Reaction Simulator. To assess driver interventions, we develop a driver reaction simulator. The simulated driver is notified when any safety alerts are raised by the ADAS (e.g., FCW) or when the driver observes any abnormalities in the vehicle's status or camera user interface (UI). These ACC abnormalities include hard braking, unexpected acceleration, the vehicle's speed exceeding the cruising speed by more than 10%, unsafe following distance with the lead vehicle (e.g., less than a vehicle length), or the mean perturbation value in the UI surpassing a noticeable threshold, with a default value of 15% representing an alert driver ($Patch.mean() > 0.15$). We assume a very alert driver who can notice any anomalies that occur within a single control cycle (10ms). A predefined emergency response will be issued by the driver accordingly (see Table 4), taking effect 2.5 seconds later (average driver reaction time). To mimic the human driver's braking behavior, we adopt a braking curve function from previous research [18].

Priority-based Control Command Dispatcher. With multiple safety mechanisms in place, there might be conflicts among the control commands issued by the OpenPilot ACC controller and those generated by the safety interventions. To resolve such conflicts, we design a command dispatcher to transmit output control commands to the CARLA actuators from various sources (e.g., ACC, AEB, simulated driver) based on their priorities, with high-priority commands overwriting low-priority ones (see Fig. 6 and Fig. 8). The simulated driver's actions have a higher priority than regular ACC outputs, and control actions from the AEB have the highest priority. The driver's actions will be executed 2.5s (average driver reaction

time) after safety alerts (e.g., FCW) or noticing other ACC malfunctions (see Table 4). ACC commands will be blocked or disengaged when AEB or driver interventions are triggered.

5 Evaluation in Simulated Environment

This section presents the evaluation of our proposed context-aware attack strategy (referred to as **CA-Opt**) using the enhanced simulation platform presented in Section 4.

5.1 Methodology

We study the following research questions by comparing the effectiveness of CA-Opt attack to several baseline attack methods in causing safety hazards (Section 5.2) and evading different safety interventions (Section 5.3):

RQ1: Does strategic selection of attack times and values increase the chance of hazards (forward collisions)?

RQ2: Does stealthiness design help maintain the attack effectiveness in the presence of safety interventions?

RQ3: Does a perception input attack achieve better performance than direct perception and control output attacks?

Baselines. We design three baseline attack strategies to answer these questions (see Table 5).

To assess the effectiveness of our optimization-based adversarial patch method in strategically selecting the attack values, we compare it to two baselines: **CA-Random**, which introduces random perturbations to perception inputs, and **CA-APGD**, which uses a state-of-the-art gradient-based method, Auto-PGD [22], to determine the perturbation values. Since the original Auto-PGD is designed for misclassification, which does not work for attacking ACC, we change the goal function (to $\nabla g(d)$, see Eq. 1) to maximize relative distance prediction. We also limit the iteration number to 5, the maximum number of control cycles (100Hz) within a perception cycle (20Hz)). Both baselines use the same context-aware method as the proposed CA-Opt attack strategy to choose the attack start time and duration. To evaluate our method's efficiency in selecting the timing and duration of attacks, we design another baseline (**Random-Opt**) that selects a random start time uniformly distributed within [5, 40] seconds and a random attack duration uniformly distributed within [0.5, 2.5] seconds. To concentrate only on the effect of different start times and durations, Random-Opt shares the same adversarial patch generation method as CA-Opt. For a fair comparison, we confine perturbations to the detected bounding box (BBox) of the LV and update BBox size and position with our proposed patch updating algorithm (Section 3.3.2).

Driving Scenarios. We model a 2016 Honda Civic, both with and without basic safety features, navigating curvy and straight sections of a highway using the "Town04_opt" map under clear weather and dry road conditions in CARLA. We simulate four high-risk driving scenarios designed based on the NHTSA's pre-collision scenario topology report [74]. The Ego vehicle, traveling at 60 mph and positioned 75 meters away, encounters a LV exhibiting various behaviors: (SC1) LV cruises at the speed of 35 mph; (SC2) LV cruises at the speed of 50 mph; (SC3) LV slows down from 50 mph to 35 mph; and (SC4) LV accelerates from 35 mph to 50 mph.

Our experiments are done on Ubuntu 20.04 LTS, with OpenPilot v0.8.9 and CARLA v9.11. A single simulation of OpenPilot contains 5,000 time steps, and each step lasts about 10 ms, which equals 50 seconds in total. However, if an attack leads to a collision, the simulation ends earlier. For Random-Opt, we randomly select ten start times and five durations for each test scenario and repeat them five times, which results in 1,000 simulations. The same total number of simulations is done for CA-Opt and other baselines.

Table 5: Overview of proposed and baseline attack strategies.

Attack	Start Time	Duration	Attack Value	#Sim.
CA-Random	Context-Aware	Context-Aware	Random	1000
CA-APGD	Context-Aware	Context-Aware	AutoPGD	1000
Random-Opt	Uniform [5,40]s	Uniform [0.5,2.5]s	Opt-based	1000
CA-Opt (Ours)	Context-Aware	Context-Aware	Opt-based	1000

Figure 9: Top: Adversarial patch examples generated using proposed method vs. random and APGD-based methods. Bottom: Success rate of CA-Opt and baseline attacks in absence of safety interventions.

5.2 Attack Success Rate in Causing Hazards

To assess the CA-Opt attack's effectiveness in inducing safety hazards (**RQ1**), we conduct experiments in the closed-loop simulation platform without enabling the safety interventions. This setting is similar to previous works [12, 16].

We consider an attack successful if a collision event is observed (Ego vehicle collides with LV) or relative distance between LV and Ego vehicle is no larger than 0m. The success rate is reported across 1,000 simulations, if not specified otherwise.

Fig. 9 displays the success rates for each attack. The CA-Random attack leads to hazards in less than 1.2% of scenarios, with an overall success rate of 0.7%. This low rate suggests that randomly generated adversarial patches minimally impact the DNN model predictions and rarely cause hazards. Increasing the random perturbation values of the adversarial patch also does not significantly enhance the success rate. This is because OpenPilot's DNN model is primarily trained to detect the front objects but not to identify their specific class, thus showing greater resilience to adversarial attacks. Additionally, patches with larger perturbation values are more visible to the driver than those from the optimization-based method (Fig. 9-Top), potentially alerting the driver to prevent hazards.

In contrast, the proposed CA-Opt attack achieves a 100% success rate in all testing scenarios, surpassing CA-Random by 142.9 times. Although CA-APGD uses a similar goal function as CA-Opt, it does not cause hazards in 46.6% of simulations, mainly due to limiting its number of iterations to satisfy real-time constraints. On the other

hand, by employing a dynamic patch updating algorithm, the CA-Opt attack ensures the continuity of the optimization process across the perception cycles and enhances the attack's effectiveness.

We also observe that the Random-Opt baseline only achieves an average success rate of 3.5%, 28.6 times lower than the CA-Opt, as it wastes resources by injecting perturbations at non-critical system states. This highlights the importance of strategic timing of attacks and the insufficiency of optimization-based methods alone in causing hazards.

Observation 1: CA-Opt is more efficient than baselines in identifying the most critical times and optimal DNN perturbation values for attacking the ACC systems at runtime and overcoming real-time constraints (C3).

5.3 Attack Stealthiness with Safety Interventions

This section studies the impact of the safety interventions and the stealthiness design on attack efficiency (**RQ2**).

5.3.1 Stealthiness in Perception Input. To evade detection by safety mechanisms and human driver, the adversarial patch should stay as stealthy as possible. Basically, the smaller the value of the pixels' perturbations, the stealthier the attack will be. Therefore, we tested our attack method with three different λ values in Eq. 1. We use two sets of metrics to evaluate the stealthiness of the patch, including (i) the degree of the pixel perturbation measured using L_2 and L_∞ distance [75] and (ii) the similarity between the original camera image and the perturbed image, calculated using RMSE and universal image quality index (UIQ) [76].

Table 6 presents the results averaged over all the test scenarios and simulations. The CA-Opt attack achieves at least a 99.2% success rate under all three stealthiness levels and keeps the perturbation degree less than 0.015 (L_∞) and 0.184 (L_2). The perturbed image with the adversarial patch has a similarity of UIQ = 0.993 (1 means identical) to the original image. We choose the λ value to be 10^{-3} in our evaluations because of its stealthiness and high attack effectiveness. Examples of the generated adversarial patches (with $\lambda = 10^{-3}$) are presented in Fig. 4 (see the zoomed-in area) and Fig. 9, which are almost invisible to human eyes.

To further evaluate the stealthiness of our attack design, we also conduct a user study with 30 participants. Results show that adversarial patches at $\lambda = 10^{-2}$ and $\lambda = 10^{-3}$ are almost imperceptible to human drivers, and the patches generated by CA-Opt attacks are less noticeable than those generated by baseline perception attacks (CA-Random and CA-APGD) (refer to Appendix D for more details).

Table 6: Attack success rate with different patch stealthiness levels.

Stealthiness Level λ	Succ. Rate	Perturbation Pixel		Image Similarity	
		L_2	L_∞	RMSE($\times 10^{-5}$)	UIQ
10^{-2}	99.2%	0.086	0.015	1.061	0.993
10^{-3}	100%	0.128	0.015	1.168	0.993
10^{-4}	100%	0.184	0.015	1.319	0.993

* L_2 and L_∞ distances are the normalized perturbation values of the attack vector matrix in the range of [0,1]. Image similarity is evaluated by comparing the RMSE and UIQ between the original image and the perturbed image with the patch. Smaller RMSE and larger UIQ mean higher similarity.

Table 7: Performance of attacks with all the safety features and different AEBS settings.

Safety Interventions	Attack Method	Intervention Activation Rate	Succ. Rate	Hazard Prevention Rate
All & AEBS Not Compromised (Independent Camera)	CA-Random	27.4%	0	100% (7/7)
	CA-APGD	100%	0	100% (534/534)
	CA-Opt	100%	**48.7%**	**51.3%** (513/1000)
All & AEBS Disabled/ Compromised (Shared Camera)	CA-Random	23.8%/ 24.3%	0	100% (7/7)
	CA-APGD	100%	0	100% (534/534)
	CA-Opt	100%	**82.6%**	**17.4%** (174/1000)

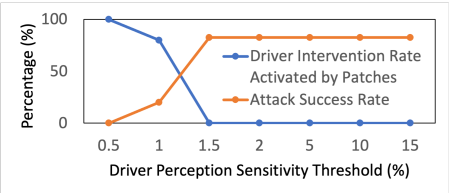

Figure 10: Evaluation with different driver sensitivity thresholds.

5.3.2 Evading Safety Interventions. For a more realistic evaluation of the effectiveness of different attack strategies, we rerun our experiments with different safety interventions (introduced in Section 4). A calibration of the safety features is performed before the experiments to ensure the interventions are triggered correctly without any false positives. We test each attack method with different AEBS configurations: (i) AEB/FCW depends on an independent camera that is not compromised, (ii) AEB/FCW utilizes compromised camera inputs similar to the ACC (simulating stock ACC and AEBS that share a camera or independent ACC and AEBS cameras that are both compromised), or (iii) AEB/FCW is disabled. Driver intervention and ACC safety constraint checking (OpenPilot Panda checks) are considered for all three settings. Here, we do not test the Random-Opt attack due to its similarity to CA-Opt but with worse performance. We assess the efficacy of each attack method using metrics such as the attack success rate, safety intervention activation rate (indicating the percentage of simulations triggering safety interventions), and hazard prevention rate (the percentage of simulations where hazards occur without safety interventions).

Table 7 shows the experimental results of each attack method with different safety intervention configurations. We observe that, regardless of the interventions, the CA-Random and CA-APGD attacks fail to cause any hazards due to their low baseline success rates (see Fig. 9) and their noticeable perturbations that trigger the driver interventions in 23.8-27.4% and 100% of scenarios. These findings highlight the effectiveness of human drivers in preventing accidents and keeping autonomous driving safe.

In contrast, with AEBS disabled, the CA-Opt attack resulted in an average attack success rate of 82.6%. We also conduct experiments that simulate higher driver sensitivity levels by decreasing the mean perturbation value threshold for activating driver intervention from the default value of 15% (see Section 4) to 10%, 5%, 2%, 1.5%, 1%, and 0.5%. As shown in Fig. 10, when perturbation thresholds are set to 0.5% and 1% (representing highly sensitive driver), the adversarial patch triggers driver interventions in all and 79.6% of the simulations, leading to attack success rates of 0% and 20.4%, respectively. However, with thresholds higher than 1.5%, our attack maintains an 82.6% success rate. This finding underscores the robustness of the generated adversarial patch in evading driver detection across a range of driver sensitivities.

When AEBS is enabled and uses the same compromised camera inputs as ACC, we observe a similar high success rate (82.6%) for the CA-Opt attacks that affect both the ACC and AEBS functionalities. However, the CA-Opt attack encounters challenges when the AEBS relies on uncompromised camera data from an independent camera. In this scenario, the attack triggers AEBS interventions in all simulations. But it still maintains a success rate of 48.7% through a gradual (stealthy) change in the vehicle state (see Fig. 11) that delays AEBS activation and leaves insufficient time for hazard prevention.

Observation 2: Our simulated safety interventions are effective in preventing accidents, and as required for L2 AVs, the human driver should always be in the loop and actively monitor ADAS to ensure safety.

Observation 3: CA-Opt attack is more effective than baselines in keeping perturbations stealthy and causing hazards without being mitigated by safety interventions.

5.4 Comparison to DNN Output and Control Output Attacks

The stealthy perturbations on the perception input can get propagated through the DNN model and ACC logic and lead to changes in the DNN output (2 in Fig. 1) and ACC control output 3. Although the attacker's goal is to maximize errors in DNN output and cause sudden accelerations on the ACC output, large deviations in vehicle states may be detected by the human driver or existing safety and defense mechanisms. To further evaluate the stealthiness of our proposed attack (CA-Opt), we compare deviations resulting from the attack to those caused by stealthy attacks directly on DNN and control outputs. Note such attacks are only possible under specific threat models (e.g., malware or wireless methods) in Table 2.

Control Output Attacks. We first examine deviations in the autonomous vehicle states and control outputs resulting from the attack compared to two baseline output attacks, called **MaxOut** and **StrategicOut**. These attacks directly modify ACC output control commands, by setting them to a maximum allowed acceleration value (MaxOut) or a strategic value (StrategicOut) based on a method from prior research [18]. But they use the same context-aware method as CA-Opt for selecting attack times and durations.

Fig. 11 illustrates an example scenario. The MaxOut attack leads to faster collisions, but also results in more noticeable changes in critical states such as gas, acceleration, and vehicle speed. These significant alterations are easily detectable by anomaly detection mechanisms or can be promptly noticed and addressed by human drivers. In contrast, fixed perturbations injected by CA-Opt attack to DNN perception inputs may not propagate to cause any changes in ACC output or if they cause any changes, it will not be larger than the maximum possible acceleration caused by MaxOut attacks. These perturbations lead to gradual deviations of system states over a longer time period, thus achieving a high success rate (as shown in Fig. 9) while reducing the likelihood of detection. Although StrategicOut produces smaller deviations strategically to avoid safety alerts, changes in vehicle states (e.g., speed) are still more noticeable than the CA-Opt perception attack.

Table 8: Performance of StrategicOut attack with all the safety features and different AEBS settings (AEBS with Shared Camera).

Safety Interventions	Attack Method	Succ. Rate	Hazard Prevention Rate
All & AEBS Activated	StrategicOut	20.3%	79.7% (797/1,000)
All & AEBS Disabled	StrategicOut	81.9%	18.1%(181/1,000)
All & AEBS Activated	OptOut	34.5%	65.5 (655/1,000)

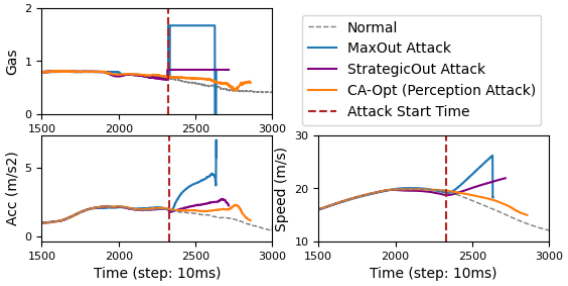

Figure 11: Context-Aware perception attacks vs. output attacks.

We also evaluate the success rate of StrategicOut attack under two different safety intervention configurations. We do not assess the MaxOut attack due to its high likelihood of being detected. Table 8 shows that without AEBS, StrategicOut achieves a success rate of 81.9% by generating attack values within safety limits and avoiding driver intervention. However, with AEBS active, using the same camera inputs as ACC, the success rate drops significantly to 20.3%, due to AEBS interventions triggered in all simulations.

We further compared the CA-Opt attack with a stealthy control output attack that causes the exact deviations of the state variables as the proposed perception attack (referred to as **OptOut**). Specifically, we reran the simulations and injected the faults by setting the control output to the recorded output traces caused by the CA-Opt perception attack. We observed that the OptOut attack achieved a higher success rate (34.5%) than the StrategicOut attack. However, it did not change the DNN predictions or affect the AEBS function, thus triggering safety interventions more easily and earlier than the CA-Opt attack. In addition, CAN outputs are encrypted in some car models [19], increasing the attack cost.

DNN Output Attacks. Similarly, we compare CA-Opt attack with a stealthy attack that directly compromises DNN output 2 (referred to as **DNNOut**) by formulating an optimization problem to maximize the RD prediction within one standard deviation while ensuring the satisfaction of safety constraints on acceleration and speed [18]. We calculate the acceleration and speed values corresponding to RD predictions by replicating the Openpilot MPC and PID algorithms. This baseline uses the same context-aware method as CA-Opt for selecting the attack times and durations. We observe that the DNNOut attack causes a more obvious change in the RD predictions (see Fig. 12) compared to CA-Opt attack on DNN inputs 1, which then results in similar obvious changes in the gas, speed, or acceleration as depicted in Fig. 11.

Observation 4: CA-Opt attack has advantage over direct DNN or control output attacks in minimizing vehicle state changes to evade detection by safety interventions, while maintaining high effectiveness in causing hazards.

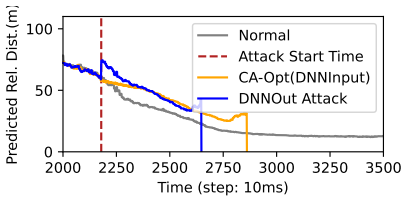

Figure 12: CA-Opt perception attack vs. DNN output attack.

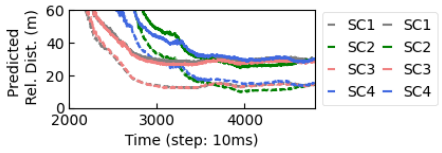

Figure 13: Relative distance predictions with (solid lines) or w/o (dashed lines) adversarial patch for different driving scenarios.

6 Evaluation in Real-World Settings

In this section we aim to answer the following questions about the effectiveness of our attack in real-world settings.

RQ4: Can our attack transfer well from simulation to real-world implementation?

RQ5: Can our attack evade detection or mitigation by the existing adversarial patch defense methods?

We also have evaluated the runtime overhead of our attack and its robustness to real-world factors, as described in Appendix F-G.

6.1 Performance on Actual Vehicles

We assess the feasibility of the CA-Opt attack on an actual vehicle (Lexus NX 2020) equipped with a production L2 ADAS, Comma 3, running OpenPilot software v0.8.9 by examining the attack impact on (i) the DNN perception module only and (ii) the end-to-end ACC.

Perception Module Evaluation. We evaluate the perception module in two scenarios of approaching an LV when (i) parked in a parking lot and (ii) driving on an actual road. In each scenario, the ACC on the Ego vehicle was tested with and without the adversarial patches injected to the camera frames.

First, the Ego vehicle was parked at distances ranging from 10m to 50m (at intervals of 5m) to the LV. We modified the OpenPilot code to display the RD predictions on the device monitor, as shown in Fig. 14-1a,1b. In these tests, the CA-Opt attack caused an average deviation of 16.2m in distance predictions, which could likely lead to a forward collision in the end-to-end ACC. This conclusion is based on our simulation experiments, where deviations exceeding 10 meters triggered sudden accelerations, leading to forward collisions.

Then, we conducted experiments using the same scenarios (SC1-SC4) outlined in Section 5.1. For an accurate assessment of the impact of the attack, we cloned the LVD's DNN model within the OpenPilot control software and ran both the original and the duplicate model on the AV simultaneously. During each perception cycle (20Hz), we initially supplied a benign image to the standard DNN model. Then, we duplicated this benign image, injected the adversarial patch to it, and then fed it to the second DNN model. The predictions from each model were recorded in separate log files. The results, presented in Fig. 13, indicate that the attack increased RD predictions by an average of 15.3 meters across all tested scenarios.

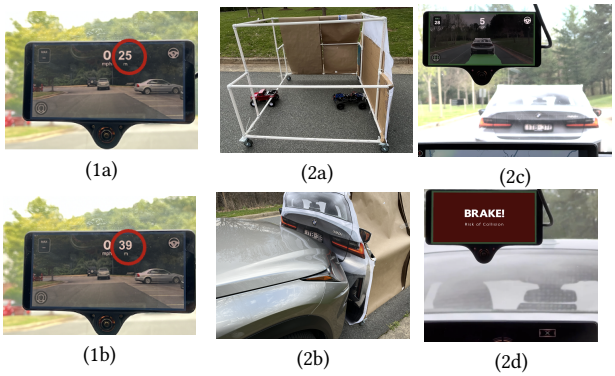

Figure 14: RD predictions w/o (1a) or w (1b) patch on an actual vehicle in a parking lot; (2a) Side view of lead car model; (2b) AV under perception attack collides with the lead car model; (2c) AV follows the car model in a benign scenario; (2d) Driver's view upon collision.

We also conducted experiments using a real-world video dataset as described in Appendix H. These experiments demonstrate the effectiveness of CA-Opt attack in impacting the DNN-based perception module in real-world driving scenarios.

End-to-End Evaluation. We also evaluate the impact of attacks on the end-to-end ACC on actual vehicle. To ensure the safety of both the driver and the vehicle, we constructed a lead car model from PVC pipe, designed to match the dimensions of a real BMW car model [77]. We aimed for the OpenPilot system to recognize this fabricated car as a genuine vehicle by attaching a rear-view image of a car to its rear end (see Fig. 14-2c). In this experiment, the AV approached the LV from a distance of 50 meters with a cruise speed set at 28 mph. Meanwhile, the LV was propelled by two remote-controlled ground robots (Fig. 14-2a). We conducted this experiment with and without attack (adversarial patches) activated.

OpenPilot software successfully recognized the lead car model as a legitimate vehicle and maintained a safe following distance and speed (about 5 mph) in the benign scenario (see Fig. 14-2c). However, in the presence of the attack, the AV continued to advance toward the lead car and eventually collided with it (Fig. 14-2b), despite AEBS being activated (Fig. 14-2d). This underscores the generalization of our proposed attack in efficiently causing safety hazards and exposes the inadequacy of existing safety mechanisms in preventing the attack. Moreover, the time between the AEBS warning and the collision was approximately 1.5-2.2 seconds, shorter than the average driver reaction time of 2.5 seconds, leaving insufficient time for a human driver to intervene and prevent the collision.

6.2 Evading Existing Defense Methods

While our proposed stealthy adversarial patches are invisible to human eye, they may be detected by some existing defense methods.

Adversarial Patch Detection. Methods such as gradient masking [78], lossy compression [79], or adversarial training [80] have been proposed for adversarial patch detection. However, these methods either need to be trained on specific attacks with high computation costs [81–83] or significantly sacrifice DNN prediction accuracy [78, 84], which negatively affect ACC systems' safety.

We assess four widely used open-source defense methods that only rely on model input transformation without the need for retraining, including adding Gaussian noise [38], JPEG compression

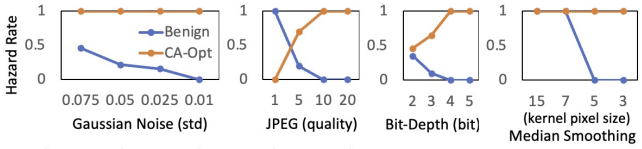

Figure 15: Results of each directly-applicable defense method.

[39], reducing image color bit-depth [40], and using spatial median smoothing [40]. We implement all these defense methods by changing each input image frame with various parameter settings (as shown in the x-axis of Fig. 15). We evaluate the attack success rates in causing hazards under each defense method while considering the effect of input transformations on the benign or attack-free image frames to maintain the baseline safety of the ACC system.

As shown in Fig. 15, JPEG compression and bit-depth reduction methods effectively reduce the attack success rate under specific parameter configurations. However, these methods fall short in maintaining the ACC's safety by leading the benign image frames to cause hazards. In instances where the benign cases do not lead to hazards, the attack hazard rate is at 100%. On the other hand, the incorporation of Gaussian noise or median smoothing reduces the ACC's LV detection accuracy. These methods are ineffective in mitigating CA-Opt attacks (hazard rate stays at 100% for all configurations), while also causing hazards for benign frames.

These results indicate that our attack can easily evade the directly applicable defense methods. More advanced methods, such as adversarial training, may need to be developed/trained specifically for our design, which are subject to future direction.

Sensor Fusion. An alternative defense against adversarial patches could involve integrating independent sensors like Lidar or radar with camera data for LVD predictions. However, Lidar is too costly for Level-2 AVs [12], and our tests found that radar-camera fusion did not prevent ACC misbehavior or collisions (see Appendix A). This may be because of using Kalman filters for sensor fusion, which assume measurement noise is zero-mean Gaussian and are vulnerable to perturbations smaller than one standard deviation of this noise [19]. In addition, sensor fusion outputs a weighted summation of radar and camera predictions, which cannot completely eliminate deviations caused by erroneous camera predictions, particularly when they significantly deviate from the ground truth. In some production ACC, camera predictions carry more weight. The sensor fusion vulnerability was also reported in previous work [85].

7 Discussion

Sim-to-Real Gap. Addressing the sim-to-real gap in AV security literature is challenging due to the risks and costs of real-road tests. In this paper, we tried to narrow this gap by developing a realistic experimental platform that integrates production ADAS control software, a physical-world simulator, and well-designed safety interventions with high-risk driving scenarios designed based on the NHTSA report [74]. Moreover, we evaluate the sim-to-real transfer possibility using an actual vehicle, a model lead car, and a publicly available dataset. However, there are still some limitations, such as using fixed models and thresholds in the design of the driver reaction simulator, that may impact evaluation results. Also, additional firmware and safety checks might be deployed in real cars, which can further limit the effectiveness of the proposed attack.

Attack Method Generalization. We demonstrate the generalization of our proposed attack on a production ACC system, OpenPilot, through closed-loop simulation, real-world AV dataset, and actual vehicle experiments. However, the vulnerability of other Level-2 production ACC systems, such as Tesla Autopilot or Cadillac Super Cruise, to our attacks remains uncertain due to their closed-source nature. While we cannot directly evaluate our attacks on these systems, it is reasonable to argue that our results hold generalization potential based on the representative nature of the OpenPilot ACC system. Specifically, our attack strategy, which leverages context awareness derived from high-level system hazard analysis, can be generalized to diverse ACC systems. Furthermore, our optimization-based attack vector generation can be applied to other DNN-based ACC systems, given the inherent vulnerability of DNNs to adversarial input perturbations [12, 13, 19, 86].

8 Related Work

Adversarial Attacks on DNN. Many works have explored the vulnerability of DNN against adversarial attacks by adding adversarial physical/digital patches or stickers [10–14, 56, 86–88]. However, most of these works focus on altering the prediction class or probability or lane line position, which do not apply to attacks against ACC. Moreover, they rely on off-line optimization of attack value, neglecting the impact of attack timing. In contrast, our work introduces a novel runtime perception attack method against DNN-based ACC systems, employing a combined knowledge and data-driven approach that considers both attack timing and value for enhanced effectiveness. The only other work on ACC [16] focused on the physical attacks without considering dynamic changes at runtime, which is not scalable to many vehicles.

Security Analysis of AVs. Great efforts have also been made in studying the security of AVs, such as the security of Lidar [89], GPS [90], radar [91], camera [92], lane detection [12, 14], multiple objects tracking [19, 27, 28], control software [18, 93], and safety mechanisms [17]. To the best of our knowledge, this paper is the first analysis of the security of Level-2 production ACC systems under stealthy safety-critical attack by considering three levels of safety interventions by constraint checking, human driver, and AEB/FCW and addressing unique challenges (Section 3.2).

9 Conclusion

This paper proposes a novel runtime stealthy attack strategy against DNN-based ACC systems, consisting of (i) a control-theoretic method for finding the most critical system contexts for launching the attacks to maximize the chance of safety hazards and (ii) an optimization-based image perturbation method for efficient generation and injection of adversarial patches to the DNN input to cause ACC control misbehavior and hazards as soon as possible before being detected or mitigated by the ADAS safety mechanisms or human driver. Experiments on a production Level-2 ADAS using an enhanced closed-loop simulation platform, a publicly available driving dataset, and an actual vehicle demonstrate the effectiveness of our approach in improving attack success rate and stealthiness compared to different baselines. This study also provides insights into the development of future ADAS that are robust against safety-critical attacks and the importance of interventions by the drivers and basic safety mechanisms for preventing attacks.

Acknowledgment

This work was partially supported by a gift from Toyota InfoTechnology Center and by the National Science Foundation (NSF) under Grants 2402941 and 1931997.

References

[1] SAE Levels of Driving Automation™ Refined for Clarity and International Audience. https://www.sae.org/blog/sae-j3016-update, 2021.
[2] Number of autonomous vehicles globally in 2022, 2022.
[3] Adaptive Cruise Control (ACC) Operating Characteristics and User Interface: Standard J2399. *Society of Automotive Engineers*, 2021.
[4] Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. *SAE international*, 4970(724):1–5, 2018.
[5] Ratheesh Ravindran, Michael J Santora, and Mohsin M Jamali. Multi-object detection and tracking, based on dnn, for autonomous vehicles: A review. *IEEE Sensors Journal*, 21(5):5668–5677, 2020.
[6] Sarthak Sharma, Junaid Ahmed Ansari, J Krishna Murthy, and K Madhava Krishna. Beyond pixels: Leveraging geometry and shape cues for online multi-object tracking. In *2018 IEEE International Conference on Robotics and Automation*, pages 3508–3515, 2018.
[7] Tesla autopilot. https://www.tesla.com/autopilot.
[8] Comma.ai. Openpilot. https://comma.ai/openpilot.
[9] Shuai Zhou, Chi Liu, Dayong Ye, Tianqing Zhu, Wanlei Zhou, and Philip S Yu. Adversarial attacks and defenses in deep learning: From a perspective of cybersecurity. *ACM Computing Surveys*, 55(8):1–39, 2022.
[10] Xin Liu, Huanrui Yang, Ziwei Liu, Linghao Song, Hai Li, and Yiran Chen. Dpatch: An adversarial patch attack on object detectors. *arXiv:1806.02299*, 2018.
[11] Mark Lee and Zico Kolter. On physical adversarial patches for object detection. *arXiv:1906.11897*, 2019.
[12] Takami Sato, Junjie Shen, Ningfei Wang, Yunhan Jia, Xue Lin, and Qi Alfred Chen. Dirty road can attack: Security of deep learning based automated lane centering under physical-world attack. In *30th USENIX Security Symposium*, pages 3309–3326, 2021.
[13] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1625–1634, 2018.
[14] Tencent. Experimental security research of tesla autopilot. *Tencent Keen Security Lab*, 2019.
[15] Juncheng Li, Frank Schmidt, and Zico Kolter. Adversarial camera stickers: A physical camera-based attack on deep learning systems. In *International Conference on Machine Learning*, pages 3896–3904, 2019.
[16] Yanan Guo, Takami Sato, Yulong Cao, Qi Alfred Chen, and Yueqiang Cheng. Adversarial attacks on adaptive cruise control systems. In *Proceedings of Cyber-Physical Systems and IoT Week 2023*, pages 49–54. 2023.
[17] Yuzhe Ma, Jon A Sharp, Ruizhe Wang, Earlence Fernandes, and Xiaojin Zhu. Sequential attacks on kalman filter-based forward collision warning systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8865–8873, 2021.
[18] Xugui Zhou, Anna Schmedding, Haotian Ren, Lishan Yang, Philip Schowitz, Evgenia Smirni, and Homa Alemzadeh. Strategic safety-critical attacks against an advanced driver assistance system. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 79–87, 2022.
[19] Saurabh Jha, Shengkun Cui, Subho Banerjee, James Cyriac, Timothy Tsai, Zbigniew Kalbarczyk, and Ravishankar K Iyer. Ml-driven malware that targets av safety. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 113–124, 2020.
[20] Xugui Zhou, Bulbul Ahmed, James H Aylor, Philip Asare, and Homa Alemzadeh. Hybrid knowledge and data driven synthesis of runtime monitors for cyber-physical systems. *IEEE Transactions on Dependable and Secure Computing*, 2023.
[21] Hongjun Choi, Sayali Kate, Yousra Aafer, Xiangyu Zhang, and Dongyan Xu. Software-based realtime recovery from sensor attacks on robotic vehicles. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, pages 349–364, 2020.
[22] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *International conference on machine learning*, pages 2206–2216. PMLR, 2020.
[23] Baidu. Apollo. https://developer.apollo.auto/.
[24] Gary Bishop, Greg Welch, et al. An introduction to the kalman filter. *Proc of SIGGRAPH, Course*, 8(27599-23175):41, 2001.
[25] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer science & business media, 2013.
[26] Richard C Dorf Robert H Bishop. *Modern control systems*. 2011.
[27] Chen Ma, Ningfei Wang, Qi Alfred Chen, and Chao Shen. WIP: Towards the Practicality of the Adversarial Attack on Object Tracking in Autonomous Driving. In *Inaugural International Symposium on Vehicle Security & Privacy*, 2023.
[28] Yunhan Jia, Yantao Lu, Junjie Shen, Qi Alfred Chen, Zhenyu Zhong, and Tao Wei. Fooling detection alone is not enough: First adversarial attack against multiple object tracking. *arXiv:1905.11026*, 2019.
[29] Adas cameras: How they work and why they need calibration. https://caradas.com/adas-cameras/.
[30] Safety Architecture. https://blog.comma.ai/how-to-write-a-car-port-for-openpilot, 2018.
[31] Li Chen, Tutian Tang, Zhitian Cai, Yang Li, Penghao Wu, Hongyang Li, Jianping Shi, Junchi Yan, and Yu Qiao. Level 2 autonomous driving on a single device: Diving into the devils of openpilot. *arXiv:2206.08176*, 2022.
[32] Supported Cars by OpenPilot. https://github.com/commaai/openpilot/blob/master/docs/CARS.md.
[33] Consumer Reports. CR Active Driving Assistance Systems: Test Results & Design Recommendations. https://data.consumerreports.org/reports/cr-active-driving-assistance-systems/.
[34] Anna Schmedding, Philip Schowitz, Xugui Zhou, Yiyang Lu, Lishan Yang, Homa Alemzadeh, and Evgenia Smirni. Strategic resilience evaluation of neural networks within autonomous vehicle software. In *43rd International Conference on Computer Safety, Reliability and Security (SafeComp)*, 2024.
[35] Comma.ai. Supercombo. https://github.com/commaai/openpilot/tree/90af436a121164a51da9fa48d093c29f738adf6a/selfdrive/modeld/models.
[36] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.
[37] Kyounggon Kim, Jun Seok Kim, Seonghoon Jeong, Jo-Hee Park, and Huy Kang Kim. Cybersecurity for autonomous vehicles: Review of attacks and defense. *Computers & Security*, 103:102150, 2021.
[38] Yuchen Zhang and Percy Liang. Defending against whitebox adversarial attacks via randomized discretization. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 684–693, 2019.
[39] Gintare Karolina Dziugaite, Zoubin Ghahramani, and Daniel M Roy. A study of the effect of jpg compression on adversarial images. *arXiv:1608.00853*, 2016.
[40] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv:1704.01155*, 2017.
[41] Cybersecurity risks for hi-tech autonomous and electric vehicles industry. https://www.linkedin.com/pulse/cybersecurity-risks-hi-tech-autonomous-electric-vehicles-samrat-seal/.
[42] Haohuang Wen, Qi Alfred Chen, and Zhiqiang Lin. Plug-n-pwned: Comprehensive vulnerability analysis of OBD-II dongles as a new over-the-air attack surface in automotive iot. In *29th USENIX security symposium (USENIX Security 20)*, pages 949–965, 2020.
[43] Rudi Mocnik, Daniel S Fowler, and Carsten Maple. Vehicular Over-the-Air Software Upgrade Threat Modelling. In *Cenex-LCV and Cenex-CAM 2023*.
[44] Abdulrahman Abu Elkhail, Rafi Ud Daula Refat, Ricardo Habre, Azeem Hafeez, Anys Bacha, and Hafiz Malik. Vehicle security: A survey of security issues and vulnerabilities, malware attacks and defenses. *IEEE Access*, 9:162401–162437, 2021.
[45] Openpilot ssh key security bypass. https://www.redpacketsecurity.com/openpilot-ssh-key-security-bypass/, 2021.
[46] Sen Nie, Ling Liu, and Yuefeng Du. Free-fall: Hacking tesla from wireless to can bus. *Briefing, Black Hat USA*, 25:1–16, 2017.
[47] Bo Luo, Mohamed Mosbah, Frédéric Cuppens, Lotfi Ben Othmane, Nora Cuppens, and Slim Kallel. *Risks and Security of Internet and Systems: 16th International Conference, CRiSIS 2021,*, volume 13204. Springer Nature, 2022.
[48] Andy Greenberg. Hackers remotely kill a jeep on the highway—with me in it. *Wired*, 2015.
[49] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *2010 IEEE symposium on security and privacy*, pages 447–462, 2010.
[50] comma connect. https://www.comma.ai/connect.
[51] Installing a fork of openpilot with workbench. https://medium.com/@jfrux/installing-a-fork-of-openpilot-with-workbench-de35e9388021.
[52] Mahmoud Hashem Eiza and Qiang Ni. Driving with sharks: Rethinking connected vehicles with vehicle cybersecurity. *IEEE Vehicular Technology Magazine*, 12(2):45–51, 2017.
[53] Sofiane Lagraa, Maxime Cailac, Sean Rivera, Frédéric Beck, and Radu State. Real-time attack detection on robot cameras: A self-driving car application. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 102–109. IEEE, 2019.
[54] Daniel Rezvani. Hacking automotive ethernet cameras.
[55] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015(S 91):1–91, 2015.
[56] Shahar Hoory, Tzvika Shapira, Asaf Shabtai, and Yuval Elovici. Dynamic adversarial patch for evading object detection models. *arXiv:2010.13070*, 2020.

[57] Amirhosein Chahe, Chenan Wang, Abhishek Jeyapratap, Kaidi Xu, and Lifeng Zhou. Dynamic adversarial attacks on autonomous driving systems. *arXiv preprint arXiv:2312.06701*, 2023.

[58] Giulio Lovisotto, Henry Turner, Ivo Sluganovic, Martin Strohmeier, and Ivan Martinovic. SLAP: Improving physical adversarial examples with short-lived adversarial perturbations. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1865–1882, 2021.

[59] Yanmao Man, Raymond Muller, Ming Li, Z Berkay Celik, and Ryan Gerdes. That person moves like a car: Misclassification attack detection for autonomous systems using spatiotemporal consistency. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 6929–6946, 2023.

[60] Chen Ma, Ningfei Wang, Qi Alfred Chen, and Chao Shen. Wip: Towards the practicality of the adversarial attack on object tracking in autonomous driving. In *ISOC Symposium on Vehicle Security and Privacy (VehicleSec)*, 2023.

[61] Openpilot: An overview and the port to the honda clarity: Hardware. https://wirelessnet2.medium.com/openpilot-an-overview-and-the-port-to-the-honda-clarity-16341d53c9aa, 2020.

[62] Abu Hasnat Mohammad Rubaiyat, Yongming Qin, and Homa Alemzadeh. Experimental resilience assessment of an open-source driving agent. In *IEEE Pacific rim international symposium on dependable computing*, pages 54–63, 2018.

[63] Saurabh Jha, Subho Banerjee, Timothy Tsai, Siva KS Hari, Michael B Sullivan, Zbigniew T Kalbarczyk, Stephen W Keckler, and Ravishankar K Iyer. Ml-based fault injection for autonomous vehicles: A case for bayesian fault injection. In *2019 49th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 112–124, 2019.

[64] Mehrdad Moradi, Bentley James Oakes, Mustafa Saraoglu, Andrey Morozov, Klaus Janschek, and Joachim Denil. Exploring fault parameter space using reinforcement learning-based fault injection. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 102–109, 2020.

[65] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328, 2017.

[66] Nancy Leveson and John Thomas. An stpa primer. *Cambridge, MA*, 2013.

[67] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[68] Richard Schram, Aled Williams, and Michiel van Ratingen. Implementation of autonomous emergency braking (aeb), the next step in euro ncap's safety assessment. *ESV, Seoul*, 2013.

[69] UN Regulation No 152 – Uniform provisions concerning the approval of motor vehicles with regard to the Advanced Emergency Braking System (AEBS) for M1 and N1 vehicles [2020/1597]. http://data.europa.eu/eli/reg/2020/1597/oj, 2020.

[70] GRVA-12-50r1e.pdf. https://unece.org/sites/default/files/2022-01/GRVA-12-50r1e.pdf.

[71] Turki Alsuwian, Rana Basharat Saeed, and Arslan Ahmed Amin. Autonomous Vehicle with Emergency Braking Algorithm Based on Multi-Sensor Fusion and Super Twisting Speed Controller. *Applied Sciences*, 12(17):8458, August 2022.

[72] Eric Shi. Openpilot: An overview and the port to the honda clarity: Hardware. https://wirelessnet2.medium.com/openpilot-an-overview-and-the-port-to-the-honda-clarity-16341d53c9aa.

[73] Panda. https://github.com/commaai/panda.

[74] Wassim G Najm, John D. Smith, Mikio Yanagisawa, and John A. Volpe National Transportation Systems Center (U.S.). Pre-crash scenario typology for crash avoidance research. Technical Report DOT-VNTSC-NHTSA-06-02, April 2007.

[75] Wikipedia. Norm. https://en.wikipedia.org/wiki/Norm_(mathematics).

[76] Zhou Wang and Alan C Bovik. A universal image quality index. *IEEE signal processing letters*, 9(3):81–84, 2002.

[77] BMW 3 Series Dimensions. https://www.carsguide.com.au/bmw/3-series/car-dimensions/2021, 2021.

[78] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. Towards the science of security and privacy in machine learning. *arXiv:1611.03814*, 2016.

[79] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *Artificial intelligence safety and security*, pages 99–112. Chapman and Hall/CRC, 2018.

[80] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv:1706.06083*, 2017.

[81] Ke Xu, Yao Xiao, Zhaoheng Zheng, Kaijie Cai, and Ram Nevatia. PatchZero: Defending against Adversarial Patch Attacks by Detecting and Zeroing the Patch. In *IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 4621–4630, January 2023.

[82] Zitao Chen, Pritam Dash, and Karthik Pattabiraman. Jujutsu: A Two-stage Defense against Adversarial Patch Attacks on Deep Neural Networks. In *Proceedings of the ACM Asia Conference on Computer and Communications Security*, pages 689–703. ACM, July 2023.

[83] Jiang Liu, Alexander Levine, Chun Pong Lau, Rama Chellappa, and Soheil Feizi. Segment and Complete: Defending Object Detectors against Adversarial Patch Attacks with Robust Patch Detection. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14953–14962. IEEE, June 2022.

[84] Chong Xiang, Arjun Nitin Bhagoji, Vikash Sehwag, and Prateek Mittal. Patchguard: A provably robust defense against adversarial patches via small receptive fields and masking. In *30th USENIX Security Symposium*, pages 2237–2254, 2021.

[85] R Spencer Hallyburton, Yupei Liu, Yulong Cao, Z Morley Mao, and Miroslav Pajic. Security analysis of camera-lidar fusion against black-box attacks on autonomous vehicles. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1903–1920, 2022.

[86] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv:1412.6572*, 2014.

[87] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[88] Zuxuan Wu, Ser-Nam Lim, Larry S Davis, and Tom Goldstein. Making an invisibility cloak: Real world adversarial attacks on object detectors. In *Computer Vision-ECCV*, pages 1–17, 2020.

[89] Takami Sato, Yuki Hayakawa, Ryo Suzuki, Yohsuke Shiiki, Kentaro Yoshioka, and Qi Alfred Chen. WIP: Practical Removal Attacks on LiDAR-based Object Detection in Autonomous Driving. In *Inaugural International Symposium on Vehicle Security & Privacy*, 2023.

[90] Junjie Shen, Jun Yeon Won, Zeyuan Chen, and Qi Alfred Chen. Drift with devil: Security of multi-sensor fusion based localization in high-level autonomous driving under gps spoofing. In *Proceedings of the 29th USENIX Conference on Security Symposium*, pages 931–948, 2020.

[91] Rony Komissarov and Avishai Wool. Spoofing attacks against vehicular fmcw radar. In *Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security*, pages 91–97, 2021.

[92] Takami Sato, Sri Hrushikesh Varma Bhupathiraju, Michael Clifford, Takeshi Sugawara, Qi Alfred Chen, and Sara Rampazzi. WIP: Infrared Laser Reflection Attack Against Traffic Sign Recognition Systems. In *Proceedings Inaugural International Symposium on Vehicle Security & Privacy*, 2023.

[93] Aolin Ding, Praveen Murthy, Luis Garcia, Pengfei Sun, Matthew Chan, and Saman Zonouz. Mini-me, you complete me! data-driven drone security via dnn-based approximate computing. In *24th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 428–441, 2021.

[94] Bowen Weng, Minghao Zhu, and Keith Redmill. A formal safety characterization of advanced driver assist systems in the car-following regime with scenario-sampling. *IFAC-PapersOnLine*, 55 no.24:266–272, 2022.

[95] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–21, 2017.

[96] John Hunt and John Hunt. Monkey patching and attribute lookup. *A Beginners Guide to Python 3 Programming*, pages 325–336, 2019.

[97] ADAS user study. https://drive.google.com/file/d/1GtMQpmgIzu4ZcjRbYKQfdE1q8WEEYPue/view?usp=sharing.

[98] Qualtrics. https://www.qualtrics.com/.

[99] Ildar Urazghildiiev, Rolf Ragnarsson, Pierre Ridderstrom, Anders Rydberg, Eric Ojefors, Kjell Wallin, Per Enochsson, Magnus Ericson, and Gran Lofqvist. Vehicle classification based on the radar measurement of height profiles. *IEEE Transactions on intelligent transportation systems*, 8(2):245–253, 2007.

[100] Harald Schafer, Eder Santana, Andrew Haden, and Riccardo Biasini. A commute in data: The comma2k19 dataset. *arXiv:1812.05752*, 2018.

A Sensor Fusion

We implement a radar sensor in the CARLA simulator and feed the data to the OpenPilot radar interface [94], to be used as an independent input by the fusion module. Specifically, we use the DBSCAN algorithm [95] to cluster the 2D point map of the relative distance and speed of perceived objects from the radar sensor and feed their mean values to OpenPilot, which are then further filtered and processed for fusion.

Fig. 16 (Top) shows an example of predictions of the relative distance to the lead vehicle from the fusion of the camera and radar measurements. We see that the radar and camera predictions agree well most of the time. Also, the error between the fusion predictions and the ground truth relative distance (based on positions of vehicles in the simulator) becomes smaller as the Ego vehicle approaches the lead vehicle (an RMSE of 0.81m after 3,000 control

cycles in the figure). Sensor fusion also helps reduce the errors in fusion predictions under attacks as shown in Fig. 16 (Bottom) and discussed in Section 6.2, even though it fails to prevent collisions in the end.

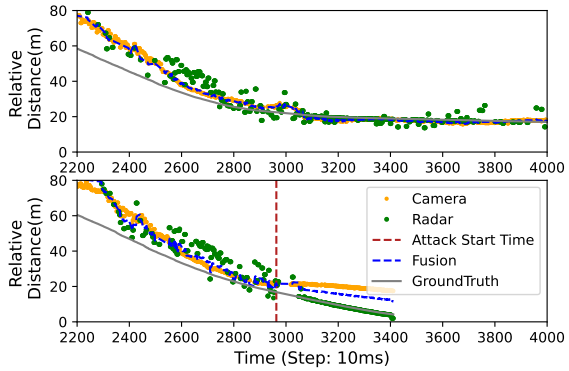

Figure 16: An example fusion of relative distance predictions based on camera and radar data compared with the ground truth under normal operation (Top) or under attack (Bottom).

B Malware Installation

Fig. 17 shows an example set of steps taken for establishing remote access and downloading a malicious code repository on a vehicle running OpenPilot, the open-source production ACC system from Comma.ai [8]. To change live camera image frames at runtime, we use a technique called monkey patching in Python, which is used for dynamically modifying or extending the behavior of an existing code at runtime or hooking a function without changing the source code[96]. For example, as shown in Fig. 17, the *send()* system library called by the camera call back function for sending the received camera frames to the DNN model can be wrapped by a malicious version *adv_send()* that implements the attack.

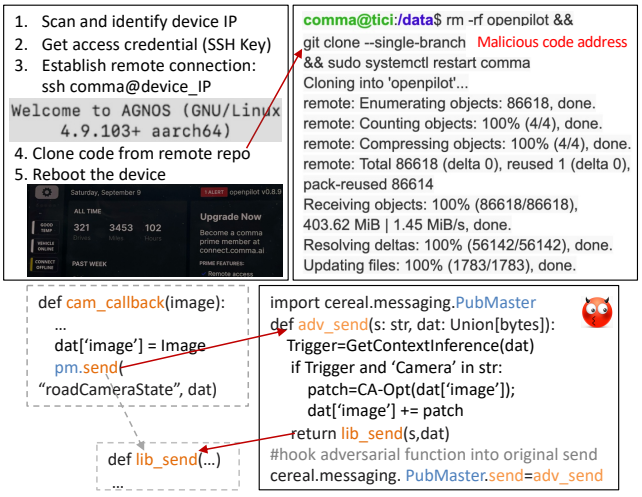

Figure 17: An example of malware installation (Top) and malicious code execution using monkey patching technique (Bottom).

C AEBS Evaluation

For a realistic implementation of AEBS in our simulation platform, we study the AEBS design of typical OpenPilot-supported car models [32]. Our AEBS design relies on the fusion of camera and radar data, as mentioned in Appendix A, aligning with AEBS design of most current OpenPilot-supported car models, as shown in Table 9.

Table 9: AEBS design in OpenPilot-supported car models.

Car Model [32]	Is AEBS using Radar/Camera/Both
Acura RDX 2018	Both
Buick LaCrosse 2019	Camera or both
Cadillac Escalade 2017	Radar or camera and ultrasonic sensors
GMC Acadia 2018	Camera and/or radar
Honda Pilot 2022	Both
Honda Ridgeline 2023	Both
Lexus ES Hybrid 2023	Both
Lexus IS 2023	Both
Toyota Avalon Hybrid 2022	Both
Toyota Camry 2023	Both

Table 10: Driving scenarios to test the AEBS with different initial distances ($Init\_dist$) between the Ego vehicle and the lead vehicle.

Lead vehicle	$Init\_dist$(m)	V_{Ego}(km/h)	V_{Lead} (km/h)
Stationary	100, 100, 150	20, 42, 58	0
Moving	100, 150	30, 58	20

Following the testing protocol specified in [69], we employ two categories and five driving scenarios (see Table 10) to assess our AEBS functionality (Section 4). Each scenario is repeated 100 times to ensure reliable outcomes. Experimental results show that in all five testing scenarios, both FCW and AEB alerts are activated, effectively preventing all hazards or collisions. On average, it takes about 1.68 seconds for AEB to stop the Ego vehicle completely.

D Stealthiness User Study

We conduct a user study [97] to further evaluate the advantages of the stealthiness design of our attack. Before recruiting participants, we secured Institutional Review Board (IRB) approval. Our study explicitly avoided collecting any personally identifying information, targeting sensitive populations, or introducing any risks to the participants. Our study included 30 participants who were asked to sit on the driver's side of an autonomous vehicle, parked in a parking lot, equipped with OpenPilot ADAS (see Section 2.3). Each participant went through different trials of pre-recorded videos displayed on the ADAS monitor and answered a series of questions after each trial using a Qualtrics survey [98]. All participants had driving experience, and 40% of them had AV driving experience.

At the beginning of the study, we provide an introduction of ADAS and present demo videos on the ADAS monitor to ensure that the participants fully understand what driving technology we are surveying.

Driving Preferences. We first ask participants to envision themselves driving this autonomous vehicle with the ADAS monitor displaying pre-recorded image frames. We inquire about how often they would look at the ADAS monitor while driving and whether alterations in the monitor's position and size influence their preference. User study results in Fig. 18 show that 99% of the participants prefer looking at the ADAS monitor during their driving experience, with 33% specifying they would do so for the majority of the driving duration. Moreover, 60% of the participants indicate a preference for a larger monitor size or a more prominent position.

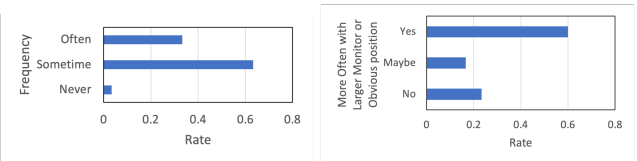

Figure 18: Results of participants' preference of looking at the ADAS monitor during driving and whether they would look more often at the monitor with a larger size or in a noticeable position.

These results indicate that the driver might notice the camera input attacks and stealthiness design might be beneficial for these attacks to evade driver intervention.

Stealthiness. We create five video sets by introducing adversarial patches into a pre-recorded highway scenario using CA-Random, CA-APGD, and CA-Opt methods with three stealthiness levels ($\lambda = 10^{-4}$, $\lambda = 10^{-3}$, $\lambda = 10^{-2}$, as detailed in Section 5.3.1). We present these videos on the ADAS monitor and ask participants whether they notice any abnormal scenarios that prompt them to assume control of the vehicle to avoid potential risk or danger. For a detailed examination, we extract an image frame from each video at the same frame index and zoom in to reveal more intricate details, followed by posing identical questions to the participants.

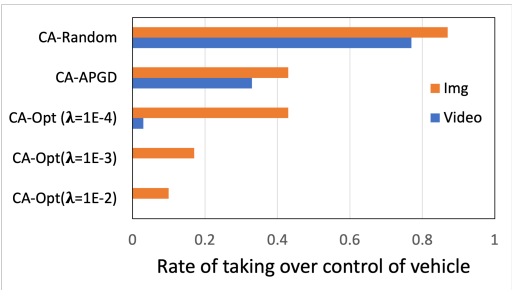

Figure 19: Results of stealthiness of each attack method.

User study results are illustrated in Fig. 19. It is evident that patches generated by the CA-Random attack are conspicuous to the majority of participants (>75%), whether observed in images or videos. In comparison, patches generated by the CA-APGD exhibit lower visibility than those produced by the CA-Random attacks. In CA-Opt attacks, the takeover rate diminishes with a rise in stealthiness level or λ value. Specifically, when λ is set at 10^{-2} and 10^{-3}, the takeover rates are below 20% for patch images and are 0% for patch videos. These findings suggest adversarial patches at $\lambda = 10^{-2}$ and $\lambda = 10^{-3}$ exhibit nearly imperceptible characteristics to human drivers, particularly in image frames when not zoomed in.

Physical Attack. We also investigate the stealthiness of physical adversarial patches as perceived by human eyes. Participants are shown an image of an adversarial patch generated through a physical attack method introduced in a prior work [16]. They are then asked identical questions. Our findings reveal a takeover rate of 80%, highlighting the inadequacy of physical patches in achieving stealthiness and evading human detection.

E Evaluation of Fake Video Attacks

To further evaluate the necessity of a stealthy patch attack, we conduct another camera attack experiment by fake video injection.

Video Recording. An Ego vehicle is configured to cruise at 40mph from 75 meters away behind a lead vehicle cruising at 35 mph in CARLA simulator. We record the image frames captured by the camera on the Ego vehicle with a duration of 50 seconds. We select a portion of the recorded image frames (7 seconds) within a straight road area to be injected at runtime.

Fake Video Attack. We rerun the simulations for each scenario introduced in Section 5.1 and replace the real-time camera frames with the selected fake video when the Ego vehicle approaches a similar position indicated by the fake video. Experimental results show that this attack causes hazards in 100% simulations and in 72.6% of simulations the Ego vehicle drives to the neighbor lane without any collisions. This is because lane lines in the fake video do not exactly overlap with the ones in the actual video.

Therefore, we compare the recorded video to the real-time image frame captured by the Ego vehicle under attacks frame by frame and select the attack start time such that the fake image frame almost matches the real-time image frame (note that this selection of perfect match at runtime attack might be impossible). We rerun the simulations and experimental results show that the perfect fake video attack achieves a success rate of 95.1% in colliding with the lead vehicle or side objects (e.g., road guard). The lower success rate of fake video attacks compared to the CA-Opt attack (100%, as shown in Fig. 9) might be due to the difference between attack start times. We do not apply the context-aware strategy to fake video attacks since it determines the attack start time dynamically at runtime, and it is challenging to select a fake image frame that perfectly matches the image frame at the time inferred by context-aware strategy at runtime.

Observation. Implementing a stealthy fake video attack is challenging as the attacker does not know the lanes the Ego vehicle will drive in the future, the positions and colors of surrounding vehicles, or the weather and road conditions. So, these differences between the fake video and the actual environment might trigger safety interventions and lead to mitigation of the attack.

Due to such differences, fake video attacks can be easily detected by existing methods that monitor the differences between two consecutive image frames. An example of image frame changes during a perfect fake video attack is shown in Fig. 20. Even if the fake videos are recorded using the same camera on the Ego vehicle driving in the same lane and weather conditions with only one lead vehicle (resembling replay attacks), an alert human driver can still notice the changes in the lead vehicle's position and size. An example of the RMSE and UIQ [76] between two consecutive image frames is shown in Fig. 21. We see that the similarity between the first frame of the fake video and the last frame of the benign video is much lower than that between other consecutive frames.

Fake Video Attack with Safety Interventions. To further evaluate the performance of fake video attacks with safety mechanisms, we rerun the experiments by launching the proposed driver

Figure 20: An example of two consecutive images before (Left) and after (Right) fake video attack.

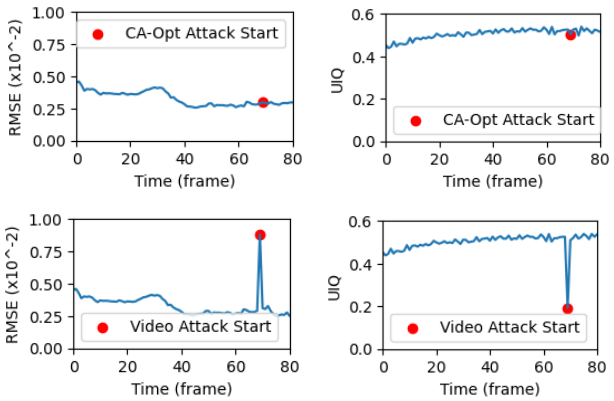

Figure 21: Similarity of two consecutive image frames with CA-Opt attack and fake video attack (starting at 69th frame) measured in RMSE and universal image quality index (UIQ) [76].

intervention 2.5 seconds (average reaction time) after the attack. Experimental results show that all the attacks are successfully prevented. Therefore, we do not further test the fake video attack while enabling AEBS and constraint checking (see Section 4).

F Robustness to Real-world Factors

To assess attack robustness, we vary front camera height based on standard passenger car profiles from manufacturers [99]. We perform our experiments with four heights between 1.1-1.7 meters and three initial distances (50m, 75m, 100m). Fig. 22 illustrates the 100% success rate of our CA-Opt attack across 12 testing scenarios. The Ego vehicle initially maintains a safe following distance, deviates from it around the 2,500-3,000 control cycle or step due to the adversarial patch, and eventually collides with the lead vehicle. These results demonstrate our attack is robust to different camera positions and initial longitudinal distances and can cause safety hazards. We also test our attacks with diverse weather (rainy, sunny, or cloudy) and lighting conditions (noon or sunset). Results show that our CA-Opt attack causes longitudinal deviations of 9.8-14.3m in the predicted lead vehicle position, while maintaining a success rate of 100% under such conditions.

G Runtime Overhead

To further assess our attack's real-world applicability, we measured its runtime overhead on a Comma3 device. We parked the Ego vehicle, equipped with OpenPilot and our attack malware, behind

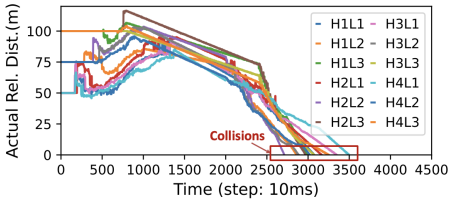

Figure 22: Actual relative distance trajectories under CA-Opt attack with different camera heights (H1:1.1m, H2:1.3m, H3:1.5m, H4:1.7m) and initial longitudinal distances to the lead vehicle (L1:50m, L2:75m, L3:100m). An actual relative distance of zero indicates collision.

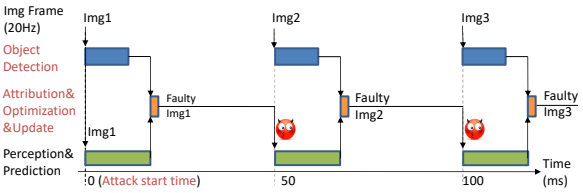

Figure 23: Runtime overhead of each step of the attack.

a lead vehicle in a parking lot, activating the ACC function with a cruise speed set to 0 mph. We record the time overhead for each component, as shown in Fig. 23, and report the average value over 5,000 control cycles.

Experimental results show that the time overhead introduced by the context inference component before activating the attacks is minimal (1.17 us). Following the activation of attacks, the time overhead for the object detection module is about 10.1 ms on average. Note that some production ADAS provide object detection and tracking features, so this overhead time could be potentially avoided. We also observe that the primary attribution algorithm [65] does not add significant overhead, leveraging gradients calculated during the patch optimization process. The total time overhead is 1.52 ms.

H Testing on a Real-world Dataset.

We perform an evaluation using the comma2k19 dataset [100], a publicly available dataset with over 33 hours of California's 280 highway commute. The dataset comprises 2019 segments, each lasting one minute, covering a 20km highway section, collected using OpenPilot hardware. From this dataset, we choose 200 videos with a clear view of the lead vehicle and a relative distance of less than 100 meters. These videos are fed into the DNN model to record predictions for relative distance (considered as ground truth). We then introduce adversarial patches, generated by different attack methods, into the videos. The manipulated videos are fed into the DNN model, and predictions for relative distance are compared with those from videos without any attacks, using metrics like average and standard deviations in the predicted longitudinal distance.

Table 11 compares the CA-Opt attack with CA-Random for different distances between the Ego and lead vehicles. CA-Random has an average deviation of 0.15m, which does not significantly impact ACC system outputs or cause hazards as the ACC system typically keeps a following distance larger than 4 meters in the absence of attacks. In contrast, when the Ego vehicle is close to the lead vehicle (less than 20m), CA-Opt achieves the highest deviation of 18.65m, showcasing the effectiveness of the proposed objective function (Section 3.3.2) in generating optimal perturbations with substantial impact.

Table 11: Performance in deviating DNN-based lead vehicle position predictions using comma2k19 dataset.

Attack	Metric	Longitudinal Deviation in DNN Prediction (m)					
		0-20	20-40	40-60	60-80	80+	All
CA-Random	Avg.	0.70	0.41	0.28	0.99	0.08	0.15
	Std	0.69	1.03	1.44	1.93	2.67	2.56
CA-Opt	Avg.	18.65	16.15	14.52	8.65	3.73	4.91
	Std	6.96	4.83	4.95	2.82	3.03	3.35