APPROVAL SHEET

This

is submitted in partial fulfillment of the requirements
for the degree of

Author:

Advisor:

Committee Member:

Committee Member:

Committee Member:

Committee Member:

Accepted for the School of Engineering and Applied Science:

[illegible]

Jennifer L. West, School of Engineering and Applied Science

ABSTRACT

Rapid advances in sensing and computing technologies have led to the proliferation of Cyber-Physical Systems (CPS). However, the increasing use of connected and complex devices, shrinking technology sizes, and shorter time to market have increased the vulnerabilities of CPS to accidental and malicious faults, posing significant challenges in ensuring their reliability, safety, and security.

This dissertation presents a holistic approach to context-aware assurance in CPS through (i) *control-theoretic* specification of safety requirements and (ii) *combined knowledge and data-driven* refinement of safety specifications for run-time safety monitoring, hazard mitigation, and design-time safety validation.

As the foundation of this research, we propose a formal framework for the specification of *safety context* defined as the combinations of the cyber-physical system states, control actions, and potential hazards, based on a control-theoretic hazard analysis method. The safety context is specified using Signal Temporal Logic (STL) to consider the timing constraints for both hazard prediction and mitigation and consists of two parts: (i) the Unsafe Control Action Specification that describes the system states under which specific control actions are potentially unsafe and can *eventually* lead to hazards at a future time; and (ii) the Hazard Mitigation Specification that identifies the control actions that if issued by the controller within a specific time period can prevent potential hazards. An optimization approach is also proposed for further refinement of the context-specific safety properties to capture the inter-scenario variability (e.g., different patient profiles or driving scenarios) and improve detection accuracy. The final context-specific safety properties are then synthesized into the logic of a safety engine that can be integrated with a CPS controller as a wrapper with only access to the input and output data, and be used for run-time context inference, hazard prediction and mitigation, and safety validation in different CPS that share the same functional specifications.

We propose combined knowledge and data-driven methods that integrate the generated safety context specifications or other safety constraints described as formal logic into machine-learning models for early hazard prediction and mitigation by enforcing the satisfaction of safety requirements while maintaining high prediction accuracy.

The generated safety context specifications can also be used for the safety validation of CPS at design time. We propose a model-driven approach orthogonal to the traditional data-driven techniques, which uses the system context specifications as the most opportune times for the activation of faults and efficiently identifies optimal fault values that can cause hazards as soon as possible without being detected by the existing safety mechanisms or mitigated by human interventions. The final goal of this approach is to discover potential design defects and safety-critical vulnerabilities in CPS to help with safety validation.

We evaluate the proposed approaches by developing open-source closed-loop testbeds that integrate real-world control software and physical-world simulators together with a fault injection engine that simulates the effect of accidental and malicious faults and real-world adverse events reported in the literature. We also evaluate our approaches using publicly available datasets or actual CPS. Experimental evaluation of the proposed assurance solutions for the case studies of artificial pancreas systems (APS) and advanced driver assistant systems (ADAS) demonstrates their generalization to a broad range of CPS with improved accuracy, timeliness, and robustness.

To my family.

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my advisor, Prof. Homa Alemzadeh, whose unwavering support, insightful guidance, and continuous encouragement have been invaluable throughout my research journey. From the initial stages of formulating my research questions to the final stages of writing and revising this dissertation, Homa has been an exemplary mentor. Her constructive feedback has always pushed me to achieve higher standards. She has been a source of personal inspiration, demonstrating how passion and perseverance are key to success in academia, and she has taught me the essential skills to become an independent and good researcher and educator.

I would also like to thank the members of my doctoral committee, Prof. Zongli Lin, Prof. Matthew Dwyer, Prof. David Evans, and Prof. John A. Stankovic, for their insightful advice, encouragement, and feedback throughout my Ph.D. work. Prof. Stankovic has been an exceptional mentor, providing meaningful feedback and guidance that has significantly influenced my research direction and career planning. His mentorship has been invaluable, and I am deeply grateful for his support and wisdom.

The completion of this dissertation would not have been possible without the help of many wonderful colleagues and collaborators. I particularly thank Haotian Ren, Maxfield Kouzel, Anqi Chen, Morgan McCarty, Prof. Cristina Nita-Rotaru, Bulbul Ahmed, Prof. James H. Aylor, Prof. Philip Asare, Chloe Smith, Anna Schmedding, Prof. Lishan Yang, Philip Schowitz, Yiyang Lu, Prof. Evgenia Smirni, Maryam Bagheri, Josephine Lamp, and Prof. Lu Feng for numerous insightful research discussions and for their support in developing and implementing the ideas in this dissertation.

I am thankful to my lab peers who accompanied me on this interesting journey, particularly Kay, Zoey, Xuren, Keshara, and Hamid, for their warm friendship and support.

I owe my heartfelt thanks to my family for all the love and support. To my parents, thank you for your endless love and unwavering belief in my abilities. You have always been there for me, providing the emotional and financial support needed to pursue my dreams. To my wonderful kids, William and Liam, thank you for being so supportive, kind, and patient. Your smiles and love have brightened my world and made me stronger and more fulfilled. And to my dear Wendy, thank you for your constant encouragement and understanding. Your sacrifice, patience, and unconditional love have been a steady source of comfort during the most demanding times of my PhD journey.

Finally, I would like to thank all those who have directly or indirectly supported me throughout this endeavor. Your contributions, no matter how small, have played a significant role in helping me achieve this milestone.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Motivation

Cyber-Physical Systems (CPS) are designed by the tight integration of software and hardware components with cyber networks and the physical world. They have been widely deployed in various applications like intelligent healthcare and autonomous driving to support safety-critical missions. However, the growing use of connected and complex devices and software, along with shrinking technology sizes and shorter time to market, has expanded attack vectors and increased vulnerabilities to accidental faults, posing significant challenges in ensuring reliability, safety, and security. This issue is evident from the rising number of reports on accidental faults and malicious attacks targeting CPS sensors, actuators, or control software, which jeopardize system operations at runtime and can lead to catastrophic consequences (e.g., fatal vehicle crashes, patient injuries, or loss of life) [3–7].

Consequently, CPS require rigorous design and thorough safety validation before deployment. Attempts have been made to advance safety validation techniques for CPS, such as sensitivity-based [8] or model-based [9] methods, verification [10], or software fault injection testing [11, 12]. But with the increasing complexity of software-intensive CPS, there is a great need for more efficient and strategic validation techniques that are applicable to the systems at scale and unified methods that can rigorously trace from informal statements about a system and its (un)desired behavior to implemented code, hardware, physical behaviors, and their interactions with the environment.

Progress has also been made in improving CPS safety and resilience using correct-by-construction techniques like quantitative risk assessment [13], control-theoretic hazard analysis [14], model-based design [15], and control synthesis [16] using formal and mathematical models. However, CPS remain vulnerable to residual faults and security vulnerabilities that might evade even the most rigorous design and validation methods and appear at run time [17]. Thus, run-time safety monitoring and hazard mitigation are essential for complementing these offline analysis and assurance methods.

Efforts have been made to ensure CPS safety using anomaly detection or online monitoring based on fuzzy logic [18], error probabilities [13], and statistical methods [19], or by developing linear or non-linear models of physical system dynamics for detecting abnormal system states and behaviors [20–23]. However, commonly used linear models cannot capture the dynamics of a complex system [22, 24]. In contrast, well-designed complex and non-linear models (e.g., [25–27]) may be difficult to derive for the human-in-the-loop CPS due to unpredictable variances in the physical world (e.g., the human body's variability in medical CPS or changing weather conditions in autonomous driving) and the changes in the behaviors and system's parameters. In addition, these approaches mostly rely on fixed and ad-hoc properties [28, 29] without the consideration of system context in the cyber layer, physical layer, as well as environmental conditions, leading to the increased possibility of false alarms. Further, these approaches do not consider the reaction time constraints for recovery and mitigation [30], resulting in the late detection and unsuccessful prevention of safety violations.

Recent works on runtime assurance in CPS leverage machine learning to enhance prediction accuracy [31], timeliness [32], or hazard mitigation efficiency [33]. Nevertheless, these models face common challenges inherent in machine learning systems, such as data limitations, lack of transparency, and performance degradation when predicting unseen data or encountering input perturbations [34–37]. Furthermore, they are frequently trained solely on data without incorporating safety guarantees, potentially leading to outputs that violate safety regulations or physical rules.

To address these limitations and ensure the safety and security of critical CPS against both accidental and malicious faults, there is an urgent need for the design of new assurance approaches in CPS that consider the system context and safety requirements with robust performance for efficient design-time safety validation and for timely and accurate runtime hazard prediction and mitigation.

1.2 Challenges

This section outlines the gaps in the state-of-the-art as well as some challenges in designing assurance approaches for safety-critical CPS, providing the motivation for the research presented in this dissertation.

1.2.1 Formal Safety Context Specification

Existing work on runtime safety monitoring and anomaly detection in CPS often rely on ad-hoc safety rules [28] or application guidelines [29] without considering the current cyber-physical system status and dynamics, leading to a large number of false alarms or missed detection [17,38,39]. Safety, as an emergent property of CPS, is context-dependent and should be controlled by enforcing a set of constraints on the system's behavior and control actions given the current system state [40]. Previous work [32, 38, 41–43] has shown that considering the multi-dimensional system context, including human, cyber, and physical systems' status, leads to improved anomaly detection accuracy and latency.

However, the system context considered in these works is specifically tailored for a particular system without a formal framework to guide the specification process, which can't be applied to other systems or applications, limiting its reusability and generalization. Recent systems-theoretic approaches to safety, like the Systems-Theoretic Accident Model and Processes (STAMP) [14], provide a way to identify unsafe context-dependent control actions that could lead to safety hazards. However, there exists a gap between the high-level safety requirements identified through these methods (e.g., STAMP [44] [45]) and the low-level formal specification of safety properties that can be used for runtime monitoring and safety assurance.

Designing a general formal framework for specifying context-dependent safety properties that can be synthesized into a safety engine for assurance purposes is critical yet challenging. This complexity arises from the need to accurately model system dynamics and thoroughly investigate the relationships between system context, control actions, and safety hazards. In addition, balancing the completeness and simplicity of the specified safety properties is challenging to ensure efficient assurance and reduced complexity.

1.2.2 Accurate and Timely Hazard Prediction

Significant efforts have been dedicated to improving the accuracy of anomaly detection, however, current methods often fall short in timely hazard identification. Typically, they detect hazards either after their occurrence or when the system states have already deviated substantially beyond a set threshold from the target region [20, 22], leading to alarms being raised too late to prevent adverse events effectively. This delayed response can significantly impact operational safety and hazard mitigation efficiency.

Moreover, the challenge lies in selecting an appropriate threshold for anomaly detection [33, 46]. A low threshold may trigger false alarms, inundating operators with unnecessary alerts and leading to alert fatigue or even unnecessary mitigation. On the other hand, setting a high threshold can increase the latency in detecting anomalies, providing insufficient time for proactive intervention. Striking the right balance between sensitivity and specificity is crucial, yet it remains a complex task due to the diverse nature of systems and environments, especially for human-in-the-loop CPS.

1.2.3 Safety Enforcement in Hazard Mitigation

Current efforts aimed at enhancing the safety and security of CPS predominantly revolve around anomaly detection and safety monitoring [20,46–48]. However, less attention has been paid to hazard mitigation strategies.

In addition to the previously highlighted challenge of accurately and timely predicting hazards, the process of devising optimal mitigation actions presents its own set of complexities. Effective mitigation strategies must not only identify and address the root causes of potential hazards but also ensure that any interventions implemented do not inadvertently introduce new risks or compromise system safety.

Furthermore, mitigating hazards in CPS environments necessitates careful consideration of various factors, including system dynamics, environmental conditions, and the specific safety requirements of the system. This entails devising mitigation actions that bring the system state back within the desired operational parameters as quickly and smoothly as possible while ensuring the satisfaction of safety requirements.

1.2.4 Efficient Safety Validation

Software fault injection has become a prevalent technique in safety validation [11,12] due to its capacity to conserve validation time and resources and uncover system vulnerabilities that may remain latent during normal operation. By deliberately introducing faults into the software, engineers can simulate various failure scenarios and assess the system's robustness under adverse conditions, such as in autonomous driving [49, 50], surgical robots [38], and artificial pancreas systems [3].

However, despite its advantages, software fault injection poses significant challenges, particularly in exploring the vast fault parameter space to identify the critical system context for the activation of fault injection as well as in generating fault values that can maximize the chance of causing safety hazards within the shortest time while avoiding

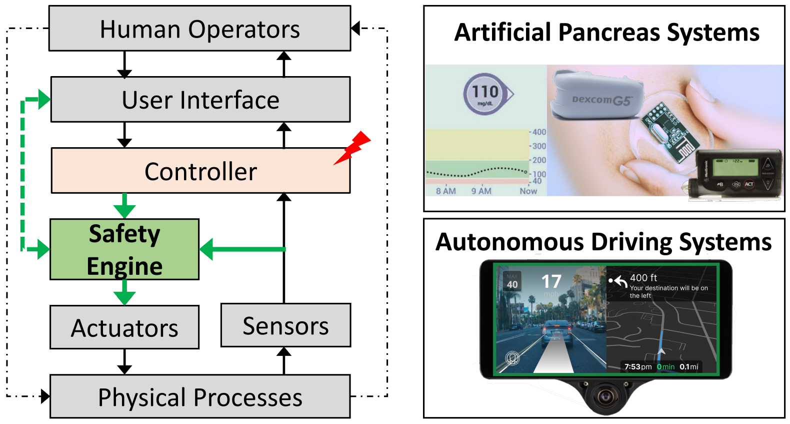


Figure 1.1: Overview of safety engine and example applications.

detection. In complex CPS, characterized by intricate interdependencies and nonlinear behaviors, pinpointing these critical parameters becomes even more challenging.

The sheer complexity and dynamism of CPS architectures contribute to the exponential growth of the state space, exacerbating the challenge of locating the specific conditions or combinations of variables that contribute to efficient safety validation. As the number of system components, interactions, and environmental factors increases, so does the complexity of the state space, making it increasingly difficult to explore all potential failure scenarios comprehensively.

1.3 Contributions

In this research, we investigate the fundamental problem of run-time assurance in safety-critical CPS. We develop a hybrid model and data-driven approach for specification, optimization, and online inference of the system safety context and understanding its relationship to unsafe control actions that lead to hazards and incidents. Note that the system context considered in this work represents the physical system states and dynamics, while the context of the environment (e.g., road or weather conditions, meals) and humans (e.g., physical activity) are beyond this scope. The generated context-specific safety specifications can be synthesized into a ***safety engine*** integrated with the control

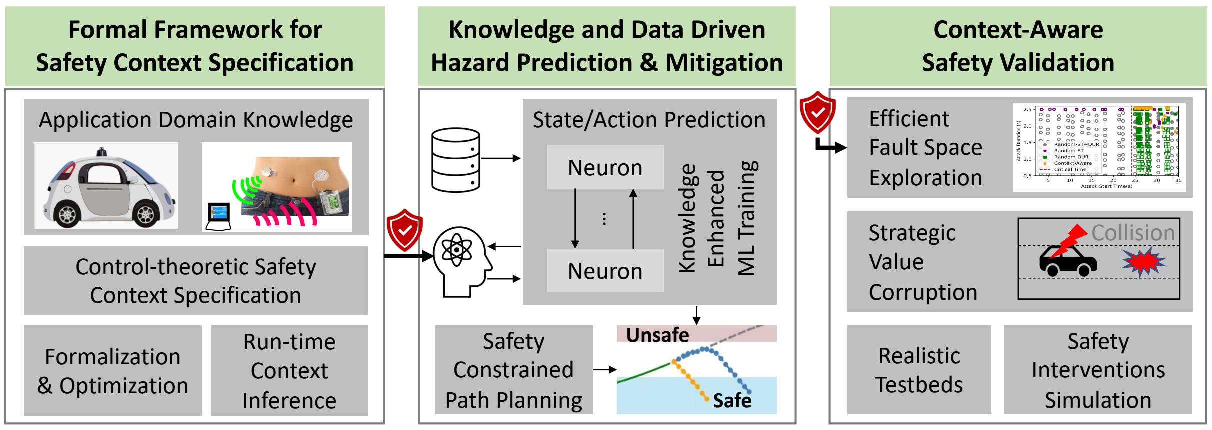

Formal Framework for Safety Context Specification
Application Domain Knowledge
Control-theoretic Safety Context Specification
Formalization & Optimization
Run-time Context Inference
Knowledge and Data Driven Hazard Prediction & Mitigation
State/Action Prediction
Neuron
Neuron
Knowledge Enhanced ML Training
Safety Constrained Path Planning
Unsafe
Safe
Context-Aware Safety Validation
Efficient Fault Space Exploration
Strategic Value Corruption
Collision
Realistic Testbeds
Safety Interventions Simulation

Figure 1.2: Overall methodology for context-aware assurance in cyber-physical systems.

software of a class of CPS with the same functional specifications (see Fig. 1.1), regardless of the internal structure or design of the controllers, to guide strategic safety validation using software fault injection during design time or to predict and mitigate potentially unsafe control actions and hazards at run-time. We focus on both accidental faults and malicious attacks targeting the CPS controller, which, upon activation, may induce errors in inputs, outputs, and the internal state variables of the CPS control software, leading to hazards or adverse events.

The main contributions of this dissertation include:

- Proposing a framework for **formal specification of safety context** for safety assurance in CPS. This framework generates template Signal Temporal Logic (STL) formulas, which can be further refined using an STL learning method and synthesized into a safety engine for runtime safety monitoring and hazard mitigation and design-time safety validation.

- Developing **combined knowledge and data driven** methods for more **accurate and timely** hazard prediction and mitigation by integrating the generated safety context specification or other safety constraints into ML models using a custom loss function, which also enforces the satisfaction of safety requirements.

- Proposing a **context-aware safety validation** strategy that can find the most critical context during an operation scenario to activate attacks or faults and strategically generates optimal fault values, with the goal of maximizing the chance of hazards and causing hazards as soon as possible before being detected or mitigated by the human operators or the existing safety mechanisms.

- Developing an open-source simulation platform using **real-world control software and physical-world simulators** for safety validation of different control algorithms as well as experimental evaluation of different safety monitors in terms of timely and accurate prediction of hazards for the case studies of an autonomous driving system (ADS) and two artificial pancreas systems (APS). Our experimental evaluations on two closed-loop APS and an ADS, publicly available datasets, and actual vehicles indicate the merits of the proposed approach in timely and accurate detection of unsafe control actions and prevention of hazards as well as efficient exploration of the fault parameter space to find potential design defects and system vulnerabilities. The implementation of the proposed approach for two different control systems (APS and ADS) also demonstrates its generalizability to different CPS and controllers (e.g., rule-based, Proportional-Integral-Derivative (PID) based [51], or Model Predictive Control (MPC) based [52]).

Fig. 1.2 shows our overall methodology for context-aware assurance in CPS. This research is organized around three main thrusts that enable the next generation of resilient CPS by advancing the state-of-the-art in run-time safety monitoring and hazard mitigation and design-time safety validation of CPS, as described in the following chapters.

1.4 Dissertation Organization

The rest of this dissertation is organized as follows. Chapter 2 introduces the formal framework for the control-theoretic specification of the safety context, which can be further refined and synthesized in a safety engine used for runtime safety monitoring, hazard mitigation, as well as design time safety validation. Chapter 3 outlines combined knowledge and data-driven approaches for optimal and secure hazard prediction and mitigation. Chapter 4 details the context-aware safety validation approach, which identifies the optimal fault timing and value, maximizing safety validation efficiency. Finally, Chapter 5 concludes the dissertation and discusses directions for future research. Appendix A presents the design and validation of closed-loop testbeds. Appendix B details the user study of the stealthiness of perception attacks.

Chapter 2

Formal Framework for Safety Context Specification

2.1 Overview

Significant progress has been made in improving CPS resilience by using correct-by-construction techniques like quantitative risk assessment [13], control-theoretic hazard analysis [14], and model-based design [15], verification [10], and control synthesis [16] utilizing formal and mathematical models. However, CPS are still vulnerable to residual faults and security vulnerabilities that might evade even the most rigorous design and verification methods and appear at run time [17]. Thus, run-time monitoring and assurance are essential for complementing such offline analysis and assurance methods.

Current techniques for run-time verification and assurance of safety properties usually depend on ad-hoc safety properties and do not consider the interactions and multi-dimensional context in the CPS. Nevertheless, as an emergent property of CPS, safety is context-dependent and should be ensured by applying a set of restrictions on the system's behavior and control actions given the current system state [32,40].

Previous research on anomaly detection in CPS demonstrated that considering the multi-dimensional system context, including the human, cyber, and physical systems' status, contributes to improved detection accuracy and latency [38,41,43]. However, the

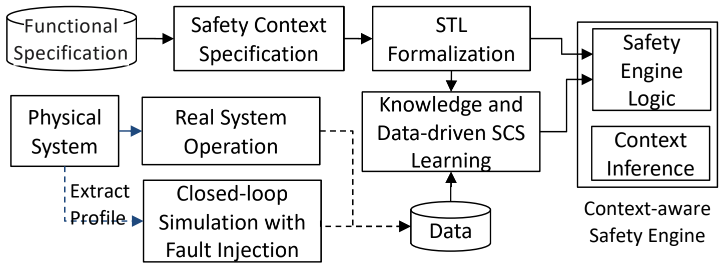
This chapter contains material from the previously published works [47,53], coauthored with H. Alemzadeh, J. Aylor, B. Ahmed, and P. Asare, copyrighted by IEEE.

lack of a formal framework for specifying safety context poses significant challenges in ensuring the effectiveness and generalization of these approaches.

Further, most previous research on CPS safety and security have focused on detecting safety-critical faults or attacks on sensor data before they reach the controller by implementing redundant hardware [54] or software [31] components, quickest change detection techniques [22], invariant monitoring [20,21], or ML-based anomaly detection [55]. However, less attention has been paid to accidental faults that directly compromise the *controller* functionality or attacks that exhibit the malicious behavior *after* the controller has received the sensor data. These attacks or faults could exploit the vulnerabilities in the communication channels [4,38], mobile and app-based controllers [56], and software development processes [57], bypass the defense mechanisms mentioned above, and expose the system and its users to potential safety hazards. In this research, we aim to address this problem by focusing on the faults and attacks targeting the controller itself while assuming the data observed by the safety engine can be protected using the above mentioned methods.

Specifically, we adopt the control-theoretic notion of system context from the STAMP accident causality model [40] and propose a formal framework for the specification and design of context-aware hazard detection and mitigation mechanisms. An optimization approach is also proposed for further refinement of the specified safety context specification (SCS), utilizing a weakly supervised STL learning method. The context-specific safety specifications are used for the automated synthesis of a safety engine that can be used for runtime safety monitoring and hazard mitigation and design-time safety validation. This chapter will mainly introduce the framework for generating and refining SCS and illustrate the process of synthesizing these specifications to design a runtime safety monitor as an example. More details on how SCS can be employed for hazard prediction and mitigation, as well as safety validation, will be provided in the next chapters.

Fig. 2.1 shows the overall framework for designing a context-aware safety engine.


Functional Specification
Safety Context Specification
STL Formalization
Safety Engine Logic
Context Inference
Context-aware Safety Engine
Physical System
Real System Operation
Knowledge and Data-driven SCS Learning
Extract Profile
Closed-loop Simulation with Fault Injection
Data

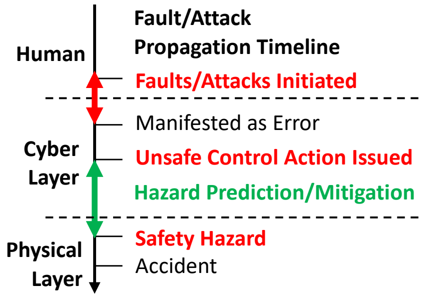


Figure 2.1: Left: Framework for the design of context-aware safety engine. Right: Fault propagation timeline.

Our approach combines the formal specification of safety context and unsafe control commands based on hazard analysis and domain knowledge with data-driven optimization techniques to generate safety properties to be checked by the run-time safety engine. We use collected data from the closed-loop CPS simulation or run-time operation to refine the safety properties with relevant parameters or to train ML models under the guidance of the generated safety specifications to improve the safety engine's timeliness, robustness, and efficiency. The safety engine is then synthesized from the safety specifications as a wrapper around the control software that only has access to the input-output interface (sensor and actuator values) and performs real-time execution of safety specification formulas for *preemptive* detection of unsafe control commands and prediction of hazards.

2.2 Model of System Dynamics

We first present the model of system dynamics used for design of our formal framework.

During each control cycle t, the CPS controller utilizes sensor measurements $x_t = (x_{1_t}, \dots, x_{n_t})$ from the continuous space $\mathbb{R}^n$ to estimate the status of the physical system and determine a control action, u_t. This action is chosen from a finite set of high-level control actions $U = \{u_1, \dots, u_r\}$, such as *Acceleration* and *Deceleration* in ADS. Each high-level control action corresponds to values of various low-level control output variables (e.g., *gas* and *brake*), which are then transmitted to the actuators. Upon execution of the control command by the actuators, the physical system transitions to a new state estimated by x_{t+1} in the state space.

2.2.1 Regions of Operation

We assume there are three mutually exclusive regions of the state space. We identify the unsafe/hazardous region $\mathcal{X}_h$ as the set of system states that lead to accidents and can be further partitioned into regions associated with particular safety hazard types H_i. The safe/target region $\mathcal{X}_*$ can be defined based on the goals and guidelines of the specific application. The set of states not included by either of these regions is referred to as the possibly hazardous region $\mathcal{X}_{*<h}$. Example regions of operation for APS and ADS are presented in Fig. 2.2. The goal of the APS controller is to keep the patient's Blood Glucose (BG) in the target range of 110-150 mg/dL, while the ADS controller's goal is to maintain a safe following distance of 2-4 seconds to the lead vehicle [58]. The unsafe regions for each example are indicated based on the definitions of hazards as later described in Section 2.7.1.

In this dissertation, we define the regions of operation similarly to previous studies [23, 59], albeit with a more conservative approach. Based on our definition of regions, the safe region and hazardous region are guaranteed, while the possibly hazardous region is approximated by minimizing the safe region while maximizing the unsafe region [60], ensuring that no unsafe control actions are overlooked.

2.2.2 Unsafe Control Actions

A sequence of *cyber control actions* $U_t = \{u_{t-k+1}, ..., u_{t-1},\ u_t\}$ issued in k consecutive control cycles are considered unsafe if upon their sequential execution in a given state sequence $X_t = \{x_{t-k+1}, ..., x_{t-1}, x_t\}$, the system will eventually transit to a state in $\mathcal{X}_h$ within the period T that U_t can affect the state space. The length of the control action sequence varies across different applications. For instance, in a robotic control system

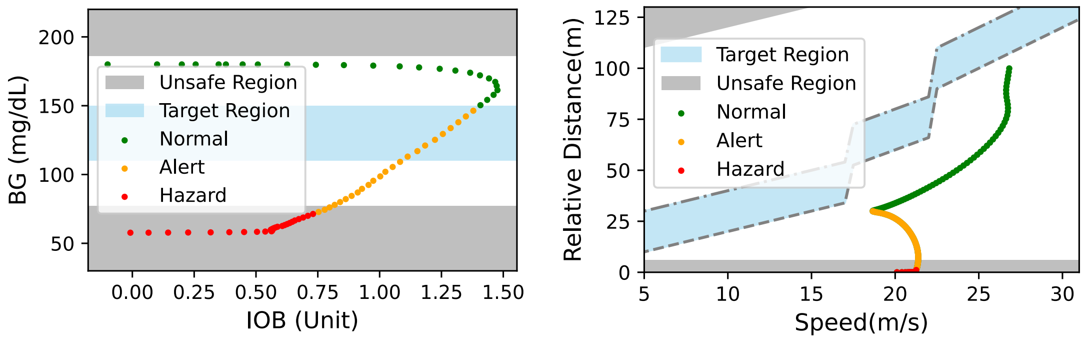


Figure 2.2: Example regions of operation: Artificial pancreas system (APS) (Left); Autonomous driving system (ADS) (Right).

with stringent real-time constraints, even a single or a few unsafe control actions might result in a safety hazard [38]. Conversely, in a slower control system like APS, a sequence of unsafe control actions may need to persist for an extended duration, possibly up to 30 minutes, to eventually lead to a hazard [47].

2.3 Safety Context Specification (SCS) Framework

We develop a formal framework for the control-theoretic specification of safety context (inspired by STAMP [40]), comprising two key components: (i) the Unsafe Control Action Specification (UCAS) that describes the system context under which specific control actions are potentially unsafe and can be used for predicting hazards or triggering safety validation; and (ii) the Hazard Mitigation Specification (HMS) that identifies one or more mitigation actions to prevent potential hazards resulting from the unsafe control actions issued by the controller.

2.3.1 Unsafe Control Action Specification (UCAS)

In order to simplify the process of defining the overall system context, we introduce the transformation $\mu(x_t) = (\mu_1(x_t), \dots, \mu_m(x_t)) \in \mathbb{R}^m$, where $\mu_i(x_t)$ represents a transformation of x_t. This transformation may involve various functions applied to x_t, such as polynomials, derivatives, or other functions, allowing for the modeling of complex combinations of state variables and their rates of change. The entire range of potential values for $\mu(x_t)$ is denoted as $\mathcal{M}$. We characterize the *system context* $\rho(\mu(x_t))$ as subsets of $\mathcal{M}$, delineated by ranges of variables within $\mu(x_t)$ that can be mapped to the regions $\mathcal{X}*, \mathcal{X}* < h, \mathcal{X}_h$. Note that the system context considered in this work represents the physical system states and dynamics, while the context of the environment (e.g., road or weather conditions, meals) and humans (e.g., physical activity) are beyond this scope.

The set of tuples $(\rho(\mu(x_t)), u_t, H_i)$ constitutes the Unsafe Control Action Specification (UCAS), denoted as $(\rho(\mu(x_t)), u_t) \mapsto H_i \subset \mathcal{X}h$. This specification specifies the system context $\rho(\mu(x_t))$ wherein issuing a control action u_t results in the system transitioning to a new context within the hazard partition H_i within the hazardous region $\mathcal{X}_h$.

The UCAS can be generated using the following steps:

1. Define the set of accidents (A) and hazards (H) of interest for the system using the control-theoretic hazard analysis method.

2. Determine the targeted transformations $\mu(x_t)$ and the sets $\rho(\mu(x_t))$ related to the hazard as comprehensively as possible based on an observable set of variables x_t. It is not necessary to determine precise thresholds for each variable that delineate each subset.

3. Enumerate all the combinations of $\rho(\mu(x_t))$ and $u_t \in U$.

4. Identify the combinations that might result in transitions to a hazardous region $H_i \subset \mathcal{X}_h$, and add tuples $(\rho(\mu(x_t)), u_t, H_i)$ into the UCAS set.

Steps 1 and 2 require manual definition based on domain expertise and input from domain experts. However, Step 3 can be automated based on the definitions established in the first two steps [45]. Similarly, Step 4 can also be automated using dynamic modeling and simulation techniques [61].

2.3.2 Hazard Mitigation Specification (HMS)

HMS is defined as a set of tuples that have the form $(\rho(\mu(x_t)), \boldsymbol{u}^\rho)$, where $\boldsymbol{u}^\rho$ is the set of safe control actions under the context $\rho(\mu(x_t))$ that lead to the transition to the safe region $\mathcal{X}_*$ and prevent hazards. The HMS can be generated through the following steps:

1. For each specified context $\rho(\mu(x_t)$ in UCAS, find all the control actions $u_t^c \in U$ such that $(\rho(\mu(x_t)), u_t^c) \mapsto \mathcal{X}_*$ and add them to $\boldsymbol{u}^\rho$, the set of safe mitigating control actions for that context.

2. Add tuples $(\rho(\mu(x_t)), \boldsymbol{u}^\rho)$ into the HMS set.

2.4 Formalization of SCS in Signal Temporal Logic

To facilitate the integration of safety assurance within CPS, we undertake the task of translating the Safety Context Specification (SCS) into a machine-checkable format. This involves converting the Unsafe Control Action Specifications (UCAS) into a set of safety properties represented in Signal Temporal Logic (STL). STL, recognized for its effectiveness in specifying temporal properties of continuous signals, provides a formal framework for articulating rigorous requirements in CPS [62,63].

By employing the bounded-time variant of STL, wherein temporal operators are associated with both upper and lower time bounds, we ensure a comprehensive representation of temporal constraints. This approach enables the synthesis of machine-checkable STL formulas, which is essential for establishing a context-aware assurance system for CPS. Through this methodology, we aim to enhance the safety assurance capabilities of CPS by providing a systematic and formal means of verifying safety properties in real-time.

The STL formula ϕ_h for a specific UCAS $(\rho(\mu(x_t)), u_t, H_i)$ is described as follows:

$$G_{[t_0,t_e]}(\varphi_1(\mu_1(x_t)) \wedge \ldots \wedge \varphi_m(\mu_m(x_t)) \wedge u_t \implies F_{[0,T]}H_i) \tag{2.1}$$

where, F represents the eventually operator $\Diamond$, and each $\varphi_i(\mu_i(x_t))$ serves as an atomic predicate, expressing an inequality on $\mu_i(x_t)$ in the format of $\mu_i(x_t) <, \leq, >, \geq \beta_i$ or its combinations. These predicates define the boundaries of each dimension $\rho(\mu i(x_t))$ within the system context $\rho(\mu_(x_t))$, with the thresholds β_i delineating the boundary values. The formula ϕ_h is globally valid (indicated by the G operator) from the start time t_0 to the end time t_e during system operation.

The UCAS for a sequence of control actions U_t, issued under a state sequence $X_t = \{x_{t-k+1}, ..., x_{t-1}, x_t\}$ spanning a window of k control cycles, is formally expressed as Φ_h:

$$G_{[t_0,t_e]}(\varphi_1(f(\mu_1(X_t))) \wedge \ldots \wedge \varphi_m(f(\mu_m(X_t))) \wedge f(U_t) \implies F_{[0,T]}H_i) \tag{2.2}$$

where, $\mu_i(X_t) \doteq \{\mu_i(x_{t-k+1}), \ldots, \mu_i(x_t)\}$ and $f(\cdot)$ represents an aggregation function such as average, Euclidean norm, or regression over k transformed measurements. When k takes the value of 1, this equation is identical to Eq. 2.1, which considers the consequences of a single control action.

Similarly, we formalize the HMS $(\rho(\mu(x_t)), u_t^c) \mapsto \mathcal{X}_*$ with the following format:

$$G_{[t_0,t_e]}((F_{[0,t_s]}(u_t^c))\mathcal{S}(\varphi_1(\mu_1(x_t)) \wedge \ldots \wedge \varphi_m(\mu_m(x_t)))) \tag{2.3}$$

which requires that $u_t^c \in \boldsymbol{u}^\rho$ should be taken within period t_s since (denoted by the S operator) the system enters context $(\varphi_1(\mu_1(x_t)) \wedge \ldots \wedge \varphi_m(\mu_m(x_t)))$. This should hold globally during the system operation.

The parameter t_s delineates the latest allowable time for the initiation of a mitigation action subsequent to the detection of a potential unsafe control action, aiming to avert hazards. This time frame is contingent upon several factors, including the context $\rho(\mu(x_t))$, the characteristics of diverse safe control actions $u_t^c \in \boldsymbol{u}^\rho$, and the operational speed of the CPS controller, and may be determined based on domain expertise and practical considerations. The specific methodology for establishing this time requirement generally lies outside the scope of this dissertation. However, an upper bound for delineating this time requirement can be derived from the estimated duration between the activation of a fault in the system and the occurrence of a hazard (referred to as Time-to-Hazard).

2.5 SCS Optimization

The parameters β_i in the STL formulas (Eq. 2.1) represent unknown boundaries, which can be inferred from either actual or simulated data using ML techniques [64] [65]. Current approaches to learning STL entail either employing classification methods with positive and negative examples, where positive instances adhere to the STL formulas and negative ones contravene them, or resorting to system simulation and experimentation to learn through the falsification of STL properties [66]. In this study, we leverage software fault injection (FI) on a closed-loop CPS to generate hazardous data traces that adhere to the STL formulas for UCAS. These traces are utilized to learn the unknown STL parameters and for adversarial training of the safety engine. Fig. 2.1 illustrates that real system operation data can also contribute to simulation model development, generation of faulty data traces, and facilitate active learning and runtime updates of the engine in practical scenarios.

We solve the problem of learning unknown thresholds β_i from a set of data traces $\mathcal{D}$ by formulating the following optimization problem:

$$\begin{aligned} &\text{minimize} \sum_{\mathcal{H}} loss(r); \;\; s.t. \\ &r = \mu_i(d(t)) - \beta_i \geqslant 0, \forall d \in \mathcal{H} : d \models \phi_h \end{aligned} \tag{2.4}$$

If the STL formula ϕ_h (Eq. 2.1) is satisfied (indicated by the $\models$ operator, which outputs a binary value from $\{True, False\}$) by a subset of hazardous traces $\mathcal{H} \subset \mathcal{D}$, the degree

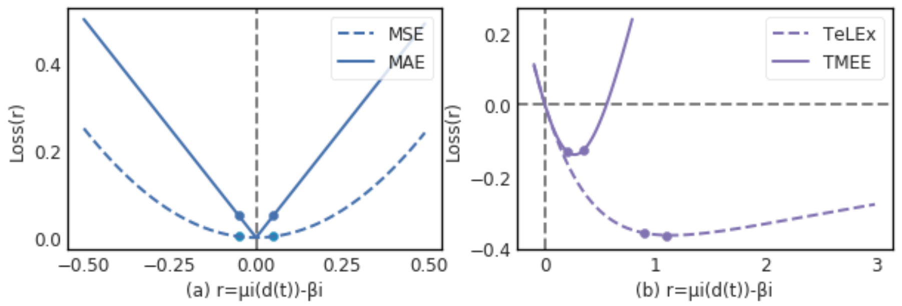


Figure 2.3: Loss functions of (a) MSE and MAE, (b) TeLEx and our proposed tight mean exponential error (TMEE) function.

of satisfaction of ϕ_h for a data trace $d \in \mathcal{H}$ at time t can be quantified using a robustness metric $r = \mu_i(d(t)) - \beta_i$ (where $\mu_i(x_t) \geq \beta_i$ represents the predicate). The objective of the optimization process is to minimize the absolute value of r as a loss function across all traces in $\mathcal{H}$ to ensure tight properties [66].

The metric resembles several commonly used loss functions in ML, such as mean squared error (MSE) and mean absolute error (MAE), typically employed for assessing parameter estimation errors. However, as depicted in Fig. 2.3a, when utilizing these loss functions, the loss values can tend to be small positive numbers near the minimum, while the actual robustness values might be small negative numbers, indicating a violation of the STL formulas. A prior study, TeLEx [66], tackled this issue by introducing a tightness function to quantify loss (Fig. 2.3b); nonetheless, the thresholds learned using such a loss function may not be sufficiently tight without manual adjustments. In this dissertation, we propose a Tight Mean Exponential Error (TMEE) loss function, as delineated below:

$$loss(r) = E[e^{-r} + r - \frac{1}{1+e^{-2r}}],\ r = \mu_i(d(t)) - \beta_i \tag{2.5}$$

which learns tight thresholds while ensuring that the faulty data traces satisfy the UCAS STL formulas.

We employed an extension of the Limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm, known as L-BFGS-B [67], which falls within the category of quasi-Newton methods for parameter estimation. Unlike conventional quasi-Newton methods [68] that directly compute the inverse of the Hessian matrix, we utilized a two-loop recursion approach [69] to estimate it. Subsequently, the L-BFGS-B algorithm leveraged the gradient of the loss function along with the estimated inverse Hessian matrix to guide the optimization process.

While our STL learning methodology shares similarities with the prior work TeLEx [66], the proposed TMEE loss function demonstrates accelerated convergence in learning unknown thresholds for the STL formulas. Notably, our preliminary experiments, involving 50 simulation runs of an artificial pancreas system controller with data sourced from a simulated diabetic patient, revealed that our optimization method achieved convergence in a significantly shorter time frame (0.02s vs. 21.79s) with substantially reduced loss values (1.15 compared to $\gg$100) in contrast to TeLEx, thereby facilitating the learning of tighter thresholds. Furthermore, the safety engine synthesized based on the tight thresholds learned through our approach exhibited higher accuracy compared to the safety rules learned using TeLEx (F1-score of 0.94 vs. 0.60).

2.6 Run-time Cyber-Physical Context Inference

The SCS STL formulas for the synthesis of the safety engine are articulated in terms of high-level and human-interpretable estimated states (e.g., Headway Time (HWT)) and control actions (e.g., *acceleration* in ADS), which may differ from the low-level sensor measurements (e.g., RADAR data) and output control commands (e.g., the amount of *gas* or *brake*) executed on the actuators.

To bridge the semantic gap between human-interpretable safety requirements and low-level measurements observed by the safety engine and to map the system's state to the STL formulas, the safety engine needs capabilities for run-time inference of both cyber and physical states. Specifically, the safety engine will infer the high-level control actions issued by the control software based on the low-level control commands sent to the actuators,

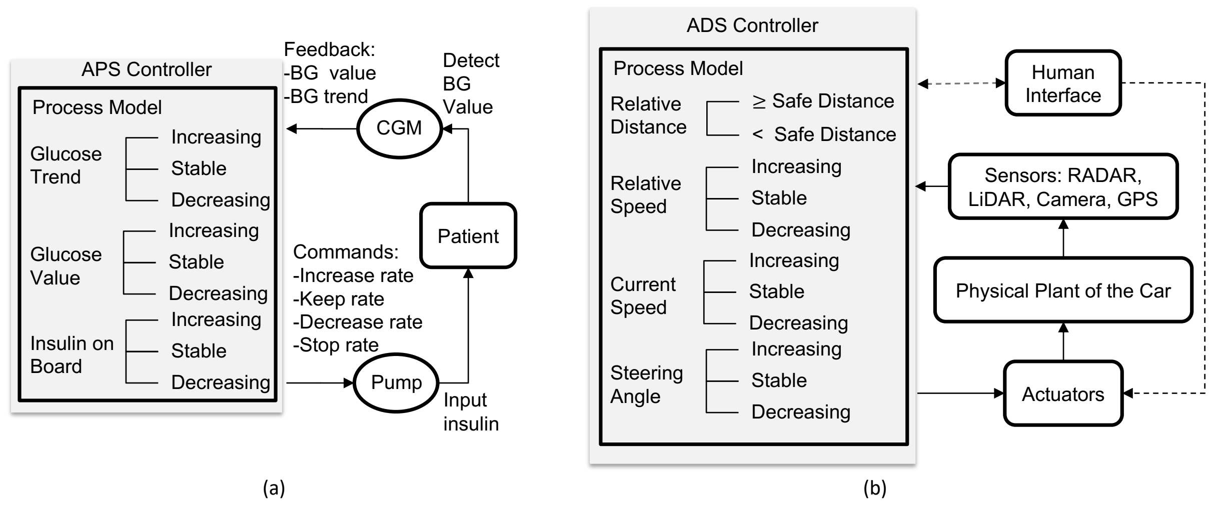


Figure 2.4: (a) Artificial pancreas system and a typical APS controller; (b) Autonomous driving system and a typical ADS controller.

and it will estimate the non-observable physical states utilized by the control algorithm based on the sensor measurements. This process can be seen as a partial replication of the controller's state estimation and control algorithms inside the safety engine.

2.7 Case Studies

To showcase the generalization and efficacy of our approach, we conducted evaluations on two distinct case studies: Artificial Pancreas Systems (APS) and Autonomous Driving Systems (ADS).

The APS controller (Fig. 2.4a) functions by estimating the current patient status (Blood Glucose (BG) value and Insulin on Board (IOB)) utilizing Continuous Glucose Monitor (CGM) readings. Subsequently, it administers the appropriate amount of insulin to the patient via a pump.

The ADS Adaptive Cruise Control (ACC) system (Fig. 2.4b) measures relative distance and relative speed to the lead vehicle using RADAR and car sensor readings. It then estimates the steering angle and brake status based on these measurements, maintaining a target following distance with the lead vehicle by issuing acceleration or deceleration control actions and adjusting gas and brake pressure accordingly.

The subsequent subsections delineate the process for generating SCS, labeling data for SCS learning, and inferring cyber-physical context for mapping measurements to SCS formulas for both of these case studies.

2.7.1 Safety Context Specification (SCS) Generation

Step 1: Initially, we identified a set of accidents and hazardous system states resulting from potential unsafe control actions issued by the controller that could precipitate accidents.

For the APS, the set of accidents (A) and hazards (H) of interest encompass:

- **A1:** Complications stemming from hypoglycemia, such as seizure, loss of consciousness, and fatality.
- **A2:** Complications arising from hyperglycemia, including tissue damage and morbidities like retinopathy, and in severe cases, death [70].
- **H1:** Excessive insulin infusion, resulting in decreased Blood Glucose (BG) levels, potentially leading to **A1**.
- **H2:** Inadequate insulin infusion, causing elevated BG levels and possibly culminating in **A2**.

For the ADS, we considered the following potential accidents and hazards:

- **A3**: Forward collision with the lead vehicle.
- **A4**: Collision with the trailing vehicle or causing traffic congestion.
- **H3**: Autonomous vehicle violates maintaining safety distance with the lead vehicle, which may result in A3.
- **H4**: Autonomous vehicle decelerates to a complete stop without a lead vehicle, which may lead to A4.

Step 2: Subsequently, we pinpointed the pertinent transformations ($\mu(x_t)$) to delineate system context. For instance, in the case of APS with sensor measurements x_t = (BG_t, IOB_t), we defined $\mu(x_t)$=(BG_t, dBG_t/dt, IOB_t, $dIOB_t/dt$), encompassing the state variables BG_t and IOB alongside their rates of change.

Steps 3-4: Following this, we compiled a list of potential UCAS for each system by identifying combinations of specific ranges in $\mu(x_t)$ and control actions (e.g., $u_t \in \{u1, u2, u3, u4\}$) that could potentially pose hazards and lead to accidents of interest (refer to Table 2.1).

Table 2.1 exhibits the ultimate safety specifications elucidated in STL formalism for both APS and ADS. For instance, the final row for APS represents a UCAS in formal representation: $(\rho(\mu(x_t)) = (BG < BGT,\ BG' < 0,\ IOB' \geqslant 0,\ IOB > \beta_{11}),\ u_4, H1)$. This specification delineates that in the system context where BG is below the target and decreasing, and IOB surpasses a certain threshold β_{11} while continuing to rise, the control action u_4 (*keep_insulin*) is deemed unsafe and is likely to result in hazard H1 if executed by the controller. This exemplifies a safety rule that can be identified or verified through consultation with domain experts. Moreover, these rules only require generation once and can be synthesized into safety engine logic, subsequently applied to diverse implementations of CPS controllers (e.g., different APS or ADS controllers) possessing identical functional specifications *regardless of the controllers' internal structure or design.*

2.7.2 Hazard Labeling for SCS Learning

For data-driven refinement and adversarial training of SCS, we necessitate examples of faulty data traces obtained from closed-loop CPS simulations or actual operations. This dataset should encompass time-series sensor measurements along with the control actions executed by the controller, labeled with the time instances when the system enters a hazardous state. It's crucial to note that an objective of the safety engine is to identify unsafe control actions and forecast these hazardous states in advance, hence the labeling method

Table 2.1: STL safety context specifications for APS and ADS.

CPS	Rule No.	STL Description of Safety Context	Implied Hazard Type
APS	1	$G_{[t_0,t_e]}((BG > BGT \wedge BG' > 0) \wedge (IOB' < 0 \wedge IOB < \beta_1) \wedge u_1 \implies$	$F_{[0,T]}H2)$
	2	$G_{[t_0,t_e]}((BG > BGT \wedge BG' > 0) \wedge (IOB' = 0 \wedge IOB < \beta_2) \wedge u_1 \implies$	$F_{[0,T]}H2)$
	3	$G_{[t_0,t_e]}((BG > BGT \wedge BG' < 0) \wedge (IOB' > 0 \wedge IOB < \beta_3) \wedge u_1 \implies$	$F_{[0,T]}H2)$
	4	$G_{[t_0,t_e]}((BG > BGT \wedge BG' < 0) \wedge (IOB' < 0 \wedge IOB < \beta_4) \wedge u_1 \implies$	$F_{[0,T]}H2)$
	5	$G_{[t_0,t_e]}((BG > BGT \wedge BG' < 0) \wedge (IOB' = 0 \wedge IOB < \beta_5) \wedge u_1 \implies$	$F_{[0,T]}H2)$
	6	$G_{[t_0,t_e]}((BG < BGT \wedge BG' < 0) \wedge (IOB' > 0 \wedge IOB > \beta_6) \wedge u_2 \implies$	$F_{[0,T]}H1)$
	7	$G_{[t_0,t_e]}((BG < BGT \wedge BG' < 0) \wedge (IOB' < 0 \wedge IOB > \beta_7) \wedge u_2 \implies$	$F_{[0,T]}H1)$
	8	$G_{[t_0,t_e]}((BG < BGT \wedge BG' < 0) \wedge (IOB' = 0 \wedge IOB > \beta_8) \wedge u_2 \implies$	$F_{[0,T]}H1)$
	9	$G_{[t_0,t_e]}((BG > BGT \wedge IOB < \beta_9) \wedge u_3 \implies$	$F_{[0,T]}H2)$
	10	$G_{[t_0,t_e]}((BG < \beta_{12}) \wedge \neg u_3 \implies$	$F_{[0,T]}H1)$
	11	$G_{[t_0,t_e]}((BG > BGT \wedge BG' > 0) \wedge (IOB' <= 0 \wedge IOB < \beta_{10}) \wedge u_4 \implies$	$F_{[0,T]}H2)$
	12	$G_{[t_0,t_e]}((BG < BGT \wedge BG' < 0) \wedge (IOB' >= 0 \wedge IOB > \beta_{11}) \wedge u_4 \implies$	$F_{[0,T]}H1)$
ADS	1	$G_{[t_0,t_e]}((HWT < \beta_{21}) \wedge (RS > 0 \wedge RS' > 0) \wedge \neg u_{22} \implies$	$F_{[0,T]}H3)$
	2	$G_{[t_0,t_e]}((HWT < \beta_{22}) \wedge (RS > 0 \wedge RS' = 0) \wedge \neg u_{22} \implies$	$F_{[0,T]}H3)$
	3	$G_{[t_0,t_e]}((HWT < \beta_{23}) \wedge (RS > 0 \wedge RS' < 0) \wedge u_{21} \implies$	$F_{[0,T]}H3)$
	4	$G_{[t_0,t_e]}((HWT > \beta_{24}) \wedge (RS < 0 \wedge RS' > 0) \wedge \neg u_{21} \implies$	$F_{[0,T]}H4)$
	5	$G_{[t_0,t_e]}((HWT > \beta_{25}) \wedge (RS < 0 \wedge RS' = 0) \wedge \neg u_{21} \implies$	$F_{[0,T]}H4)$
	6	$G_{[t_0,t_e]}((HWT > \beta_{26}) \wedge (RS < 0 \wedge RS' < 0) \wedge u_{22} \implies$	$F_{[0,T]}H4)$

* BGT: BG target value; $BG' = dBG/dt$, $IOB' = dIOB/dt$;
* HWT: Headway Time = Relative Distance/Current Speed [71]; RS: Relative Speed = Current Speed - Lead Speed; $RS' = dRS/dt$;
* $u_{1,2,3,4}$:decrease_insulin, increase_insulin, stop_insulin, keep_insulin;
* $u_{21,22}$: Acceleration, Deceleration; t_0, t_e: start time and end time of the simulation.

employed for hazards cannot be directly utilized by the safety engine. In this study, we adopt an automated labeling approach utilizing common objective metrics endorsed by the research community and commonly practiced guidelines. These metrics and guidelines will be detailed in the subsequent description.

For the APS case, we employed the concept of the Risk Index (RI) [72, 73], which encapsulates both the glucose variability and the associated risks of hypo- and hyperglycemia, to label the data. We computed the low blood glucose index (LBGI) and high blood glucose index (HBGI) for a data trace $\mathcal{D}$ of blood glucose (BG) readings using the following equations:

$$risk(BG) = 10 * (1.509 * [(ln(BG))^{1.084} - 5.381])^2 \tag{2.6}$$
$$LBGI = 1/n \sum_{\mathcal{D}} risk(BG); \text{for each } BG < 112.517$$
$$HBGI = 1/n \sum_{\mathcal{D}} risk(BG); \text{for each } BG \geqslant 112.517$$

We deemed a window (e.g., one hour) of blood glucose (BG) readings as hazardous if the risk indices exceeded a high-risk threshold (e.g., $LBGI > 5$ and $HBGI > 9$, as defined by previous studies [73, 74]) and exhibited a persistent increase, suggesting a heightened probability of hypo- or hyperglycemia.

For ADS, we label the data points as hazardous if the relative distance between the autonomous vehicle and the leading vehicle is non-positive or the autonomous vehicle decelerates to a complete stop with a considerable relative distance (e.g., greater than 100 meters [75] that is the range of a medium-range radar [76]) to the leading vehicle, which might lead to a potential collision with the trailing vehicle or causing congestion.

2.7.3 Context Inference for SCS Matching

For APS, the state variable Insulin On Board (IOB) might not be directly observable by the safety engine. Consequently, we need to derive its value from a sequence of insulin rate history. Following the injection of insulin into a patient's body, the IOB gradually increases, reaching its maximum level at t_{peak} (e.g., 75 minutes), after which it begins to decrease until it reaches zero. We calculated the IOB before the peak time using the equation proposed by [77]:

$$IOB(t) = I(t_0) * [-k_1(0.2(t - t_0) + 1)^2 + k_1(0.2(t - t_0) + 1) + 1] \tag{2.7}$$

and derived the IOB between peak time and the end of the duration of insulin action using the following equation:

$$IOB(t) = I(t_0) * [k_2(t - t_0 - t_{peak})^2 - k_3(t - t_0 - t_{peak}) + 0.55556] \tag{2.8}$$

where k_i are coefficients. The accumulated IOB under the effect of an insulin rate sequence is the integral of IOB calculated using the above equations.

For ADS, we estimate the high-level state variables, headway time, and relative speed, based on the current speed of the autonomous vehicle and the relative distance between the autonomous vehicle and the leading vehicle, measured by low-level car sensors such as GPS and Radar.

In both case studies, the safety engine translates the low-level control commands issued by the controller into the high-level control actions described in SCS by computing the rate of change in a window of measurements. For instance, an *increase_ insulin* or *decrease_ insulin* control action can be detected by calculating the slope of insulin samples.

2.7.4 Hazard Mitigation and Recovery

Upon detection of an unsafe control action issued by the controller, the safety engine initiates mitigation measures to prevent potential hazards. These measures involve correcting the command, regardless of whether its value falls within the acceptable range, and delivering a new command ($u_t^c \in \boldsymbol{u}^\rho$) to the actuator. For instance, if the insulin dosage is excessive, it can be reduced, or if it is insufficient, additional insulin can be administered. This corrective process continues until the system returns to a safe state and the safety engine ceases to raise alerts. Developing a mitigation mechanism with a high recovery rate while minimizing the introduction of new hazards is a complex challenge. Algorithm 1 outlines one potential implementation of a mitigation algorithm for preventing hazards in APS. Further exploration of mitigation algorithms, incorporating formal specification and learning from simulation data, will be discussed in Chapter 3.

Algorithm 1: Hazard Mitigation Algorithm

```
1  Mitigate ← 0
2  while $t < t_e$ do
3      $\mu(x_t) \leftarrow (BG_t, IOB_t, BG_t^{'}, IOB_t^{'})$
4      $u_t, u_t^c \leftarrow u_i \in \{u1, u2, u3, u4\}$
5      if $\rho(\mu(x_t)) \in \mathcal{X}_*$ then Mitigate ← 0, continue
6      for $\phi_i$ in STL of UCAS do
7          if $(\rho(\mu(x_t)), u_t)$ matches $\phi_i$ then
8              Mitigate←1, Hazard ← $H_i \in \{H1, H2\}$
9      end
10     if Mitigate == 1 then
11         if Hazard == $H_1$ then $u_t^c \leftarrow 0$
12         else if Hazard == $H_2$ then $u_t^c \leftarrow f(\rho(\mu(x_t)), u_t) \in \boldsymbol{u}^\rho$
13 end
```

$f(.)$ describes a context-dependent function for selecting the mitigation action. In our experiments, we instead use a fixed maximum value of insulin to enable a fair comparison with baseline non-context-aware safety engines.

2.8 Experimental Evaluation

This section presents the experiments and results of evaluating and comparing the proposed context-aware monitors designed using the SCS learning approaches detailed in Section 2.5. Specifically, we focus on the STL optimization with threshold learning method, denoted as CAWT (Context-Aware With refined Thresholds).

2.8.1 Closed-loop Cyber-Physical Simulation Platforms

To facilitate our evaluation, we developed an open-source simulation environment[1], as depicted in Fig. 2.5, which integrates closed-loop simulations of two exemplary APS and ADS control systems with a software Fault Injection (FI) engine. This environment enables us to assess various safety monitors effectively.

2.8.1.1 APS Testbeds

We integrated two widely-used APS controllers, namely Proportional-Integral-Derivative (PID) [51] based OpenAPS [77] and rule-based Basal-Bolus [78], with two distinct patient

[1][Available Online: https://github.com/UVA-DSA/CPS-Runtime-Monitor]

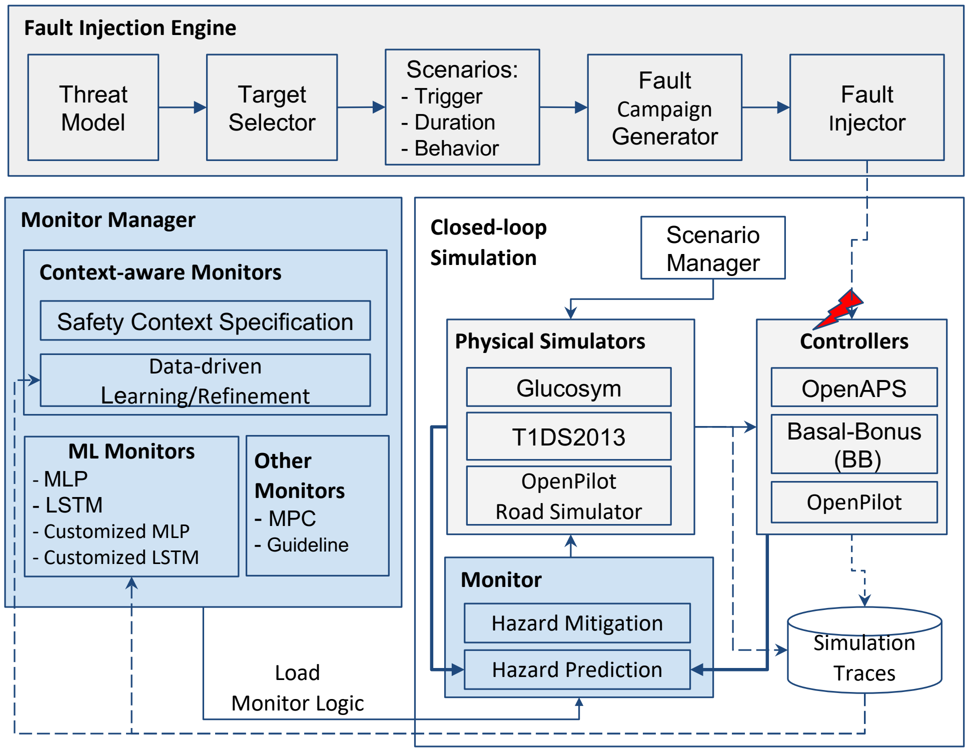


Figure 2.5: Experimental setup for evaluating various safety monitors, integrating three closed-loop simulation platforms: Glucosym simulator with OpenAPS controller, UVA-Padova T1DS2013 simulator with Basal-Bolus controller, and OpenPilot road simulator and controller, along with a software FI engine.

glucose simulators: Glucosym [79] and the UVA-Padova Type 1 Diabetes Simulator [26]. The Glucosym simulator incorporates models of 10 actual Type I diabetes patients, while the UVA-Padova Type 1 Diabetes Simulator S2013 (T1DS2013) features 30 virtual patients representative of the Type 1 Diabetes Mellitus (T1DM) population observed in a clinical trial [80]. Moreover, T1DS2013 has FDA approval for pre-clinical testing of APS [26, 81]. More details about the APS testbed design are provided in Appendix A.1. Our closed-loop testbed underwent validation using real data from a clinical trial (see Appendix A.1.5), ensuring that the simulated data meets the requirements of relevance, completeness, accuracy, and balance [82] for the development of ML models [3]. An example of the architecture of closed-loop simulation of OpenAPS with a glucose simulator is also shown in Fig. 2.6.

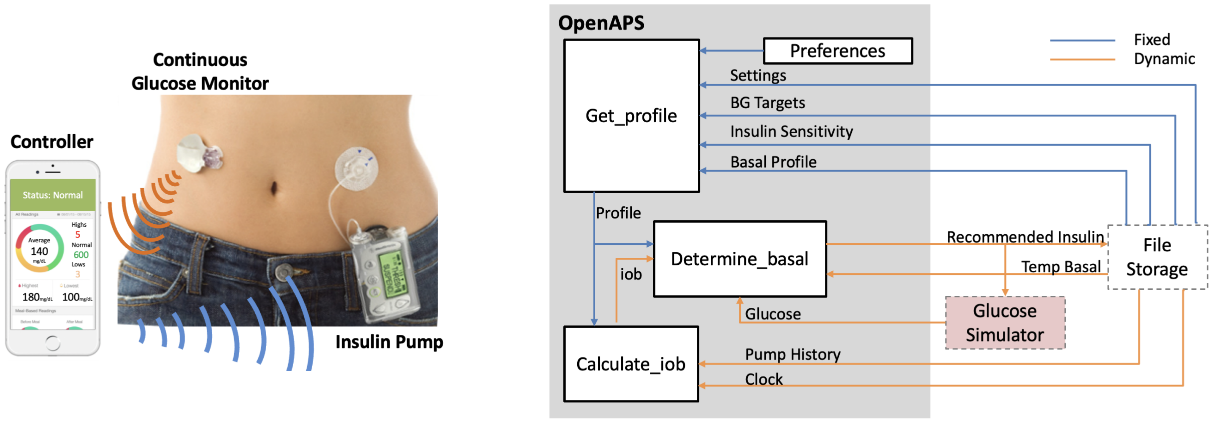


Figure 2.6: Left: Artificial pancreas system; Right: Closed-loop simulation of OpenAPS and a glucose simulator.

2.8.1.2 ADS Testbed

Fig. 2.7 presents the developed basic realistic testbed for closed-loop simulation of ADS, integrates an open-source ADAS control software, OpenPilot from Comma.ai [83] and the state-of-the-art physical-world driving simulator, CARLA.

OpenPilot. We utilized OpenPilot as our control software as it is the only open-source alpha-quality commercial driving agent. OpenPilot has been deployed in real cars on the road by over 10,000 active users. It provides adaptive cruise control (ACC) and automated lane centering (ALC) capabilities to more than 250 supported car makes and models, such as Honda Civic 2016-2023 and Ford Explorer 2020-2023 [84]. This functionality is achieved using additional hardware, the Comma 3X [85], which can control the gas, brake, and steering.

CARLA Simulator. CARLA is built for flexibility and realism for rendering and physics simulation, implemented as an open-source layer over Unreal Engine 4 [86], which provides state-of-the-art rendering quality, realistic physics, basic NPC logic, and an ecosystem of interoperable plugins. In addition to open-source code and protocols, CARLA also provides open digital assets (e.g., urban layouts, buildings, vehicles), which allows us to control the motion of multiple objects for the design of any complex scenarios. Further, CARLA can update vehicle states by executing a control command issued by the

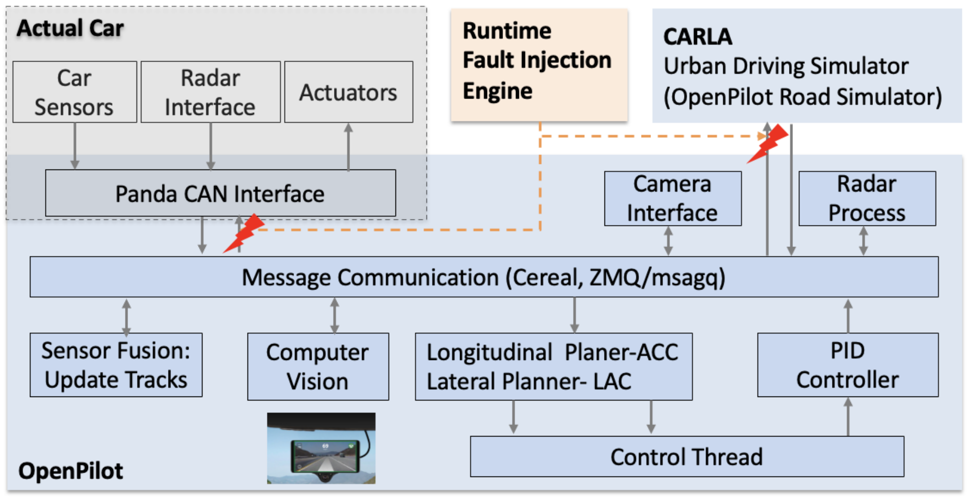

Actual Car
Car Sensors
Radar Interface
Actuators
Panda CAN Interface
Runtime Fault Injection Engine
CARLA Urban Driving Simulator (OpenPilot Road Simulator)
Camera Interface
Radar Process
Message Communication (Cereal, ZMQ/msagq)
Sensor Fusion: Update Tracks
Computer Vision
Longitudinal Planer-ACC Lateral Planner- LAC
PID Controller
Control Thread
OpenPilot

Figure 2.7: Basic closed-loop ADS testbed.

human driver of ADAS in the physical world, helping evaluate the attack effect on the vehicle at runtime without manual shifting or rotating objects in the image frame, which is required for testings on recorded videos.

We ran the experiments with both APS controllers and simulators as well as OpenPilot (v.0.4.2) on an x86_64 PC with an Intel Core i9 CPU @ 3.50GHz and 32GB RAM running Linux Ubuntu LTS. We used TensorFlow v.2.5.0 to train our ML models.

2.8.2 Scenario Simulations

In the APS simulations, we conducted experiments with initial Blood Glucose (BG) values ranging from 80 to 200 mg/dL for 150 iterations, representing 12.5 hours in an actual APS control system, without add-on meals. This setup mimics a patient's routine of eating dinner, going to sleep, and having the next meal the following day after the simulation period. Additionally, we evaluated our approaches across 20 different patient profiles, with 10 patients in the Glucosym simulator and 10 in the T1DS2013 simulator, to account for possible inter-patient variability.

In the ADS assessment, the OpenPilot simulator and controller were executed for 150 iterations, with each iteration simulating 200ms of real road driving. We simulated four

Table 2.2: Simulated fault and attack scenarios.

Type	**Approach**	**Simulated Scenario**	**Representative FDA Recalls**	**Possible Adverse Events**
Truncate	Change output variables to zero value [88] [89]	Availability attack [90]	Z-1074-2013 [1] Z-1034-2015	Device Malfunction/ Hypoglycemia/ Hyperglycemia/ Injury [91]/ Death [92]
Hold	Stop refreshing selected input OR output variables [22] [89]	DoS attack [93, 94]	Z-1359-2012 Z-0929-2020	
Max/Min	Change the value of targeted variables to their maximum or minimum allowed values [22] [49]	Integrity attack [88]/ Memory fault	Z-1562-2020 Z-2165-2020	
Add/Sub	Add or subtract an arbitrary or particular value to or from a targeted variable [22] [95]			

[1] Recall IDs assigned by FDA which can be searched for on `https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfres/res.cfm`.

driving scenarios classified as high-risk in the pre-collision scenario topology report by the National Highway Traffic Safety Administration (NHTSA) [87]:

- The lead vehicle is driving at a constant speed (40mph).
- The lead vehicle accelerates and then slows down.
- The lead vehicle slows down and then accelerates.
- The lead vehicle slows down to a complete stop.

2.8.3 Fault Injection Experiments

We collected experimental data from closed-loop CPS simulations with FI for adversarial training and testing of the proposed safety monitor and other baseline monitors.

Threat Model: We assume that both accidental faults and malicious attacks, resembling those reported for CPS (see Table 2.2), have the potential to target the CPS controller. Upon activation, these faults or attacks may induce errors in inputs, outputs, and the internal state variables of the CPS control software, leading to the hazards outlined in Section 2.7.1 and resulting in adverse events. Regarding malicious attacks, we assume that attackers have gained unauthorized remote access [96] to a CPS control system through various means, including stolen credentials [97], exploitation of vulnerable services [98], or insider attacks involving network penetration [5, 38]. Such attacks may exploit the network to which the target CPS controller is connected. Even in the absence of network connectivity, attackers might exploit physical access points such as USB ports or Bluetooth connections to infiltrate the target device and deploy malware. Table 2.2 provides examples of such fault and attack scenarios, along with vulnerabilities in the control system that have resulted in real recalls and potential adverse events.

We developed a source-level FI engine that directly perturbs the values of the controller's state variables within their acceptable ranges over a random period of time to simulate the effect of such fault and attack scenarios. We assume that errors are transient and only occur once for a specific duration per simulation. For each FI scenario shown in Table 2.2, the FI engine determines (i) the target state variable, (ii) the error value to inject, (iii) the trigger condition of the error, and (iv) the duration of the injected fault.

To diversify the fault injection process, we randomly select start times and durations for injecting faults. This approach resulted in a total of 882 and 1200 fault injections for each patient and driving scenario, respectively. Consequently, we amassed a total of 2,646,000 simulation samples used for training and testing different monitors. We adopted a 4-fold cross-validation setup for both threshold learning and the evaluation of our context-aware safety monitors, as well as for model training and testing of the baseline ML monitors.

To assess the efficacy of the safety monitor against adaptive adversaries, we also contemplate a more formidable attacker equipped with all the necessary knowledge about the target controller and safety monitor, enabling them to execute specific types of stealthy attacks (refer to Section 2.8.6.5). However, conducting a comprehensive evaluation of the proposed approach against all conceivable stealthy attacks (e.g., replay, zero-dynamics, pole-dynamics, and covert attacks [99, 100]) falls beyond the scope of this work.

Table 2.3: Rules of medical guideline monitor.

No.	Description
1	$\phi 1 = \Box(BG > 70) \wedge (BG < 180))$
2	$\phi 2 = \Box((\Delta BG > -5) \wedge (\Delta BG < 3))$
3	$\phi 3 = ((BG < \lambda_{10}) \Rightarrow \Diamond_{[0,\alpha]}(BG > \lambda_{10}))$
4	$\phi 4 = ((BG > \lambda_{90}) \Rightarrow \Diamond_{[0,\alpha]}(BG < \lambda_{90}))$

2.8.4 Baseline Monitors

In order to assess and contrast the performance of the proposed context-aware monitor, CAWT, in accurately and promptly predicting hazards, we crafted several baseline monitors representative of the current state-of-the-art safety monitoring and defense approaches for CPS.

2.8.4.1 Medical Guidelines Monitor

We crafted a baseline safety monitor, denoted as Guideline, based on generic medical guidelines proposed in [29], without taking into account patient characteristics or control software. The safety rules of the Guideline monitor are shown in Table 2.3. The Guideline monitor issues alerts when the Blood Glucose (BG) value falls outside the normal range of [70, 180] mg/dL, experiences a sharp change, or remains below its tenth percentile λ_{10} or above its ninetieth percentile λ_{90} for an extended safe period (e.g., 30 minutes).

2.8.4.2 Model Predictive Control Monitor

We formulated two baseline monitors based on Model Predictive Control (MPC) [16,101], a widely used technique in process control systems, for both APS and ADS.

For APS, the MPC monitor predicts the potential Blood Glucose (BG) value (BG_{t+1}) following the execution of the pump's command (I_t) on the patient's current state (BG_t)

using the Bergman and Sherwin model [102]:

$$dBG(t)/dt = -(GEZI + I_{EFF}) * BG(t) + EPG + R_A(t) \tag{2.9}$$

where, $GEZI, I_{EFF}, EPG$ are patient-specific parameters, and $R_A(t)$ is glucose appearance rate. An alarm will be generated if the predicted BG value goes beyond the patient's normal range (same as the medical guidelines).

In developing the MPC monitor for ADS, we employed the following dynamic model of the vehicle [103]:

$$\begin{aligned} dv(t)/dt = 3.33 * Gas(t) * P_{peak}/m/v(t) - 3 * Bk(t) \\ - (0.01g + 0.15v^2(t)) - GD + CRP(t) \end{aligned} \tag{2.10}$$

where, $v(t)$ denotes the current speed of the vehicle, while m, $Gas(t)$, and $Bk(t)$ respectively represent the vehicle mass, the output of the gas, and the brake. Additionally, P_{peak} signifies the peak power, g denotes the gravitational force of Earth, GD describes the road grade, and $CRP(t)$ characterizes the impact of creep force, which is contingent upon $v(t)$. The baseline monitor triggers an alert if the vehicle is projected to decelerate at a rate of 3 m/s^2 or more for a duration of at least one second [104].

2.8.4.3 ML-based Monitors

We utilized two state-of-the-art machine learning approaches, multi-layer perceptron (MLP) and Long-short-term memory (LSTM), to train two baseline monitors. In particular, we framed the task of detecting an unsafe control action as a context-specific conditional event, outlined as follows:

$$y_t = p(\exists t' \in [t, t+T] : x_{t'} \in \mathcal{X}_h | \bar{X}_t, \bar{U}_t) \tag{2.11}$$

When presented with an issued control action $\bar{U}_t$ at the current system state $\bar{X}_t$, represented by the average values of U_t and X_t respectively, the ML model produces a binary output y_t that categorizes U_t as safe or unsafe. We employed a fully connected two-layer MLP architecture, consisting of 256 and 128 neurons respectively, followed by a fully connected layer with ReLU activation, and finally a softmax layer to derive the hazard probabilities.

During the training phase, we assigned a positive label to y_t if any hazard occurred within a defined time window (e.g., the duration of insulin action for APS) after the sequential execution of U_t. We classified a simulation data trace as hazardous if any sample within it was marked as unsafe.

For individual patients or autonomous vehicles, we trained the model using eighty percent of the data traces, preserving the time sequence of samples within each data trace. We set a validation split rate of 0.1 and reserved the remaining twenty percent of the dataset for testing. To assess the overall performance of the final ML model, we employed 4-fold cross-validation.

Additionally, leveraging its ability to capture temporal dependencies in time-series data, we employed an LSTM model as a baseline monitor. This model was trained using input data $X_t = \{x_{t-k+1}, ..., x_{t-1}, x_t\}$ and $U_t = \{u_{t-k+1}, ..., u_{t-1}, u_t\}$ with a sliding time window of k.

$$y_t = p(\exists t' \in [t, t+T] : x_{t'} \in \mathcal{X}_h | X_t, U_t) \tag{2.12}$$

We experimented with various model architectures and identified the optimal configuration as a two-layer stacked LSTM with 128-64 units. The input time steps were set to 30 minutes for APS and 1 second for ADS. Both the LSTM and MLP models were trained using the Adam optimizer [105], employing the sparse categorical cross-entropy loss function and a learning rate of 0.001. Additionally, we incorporated a dropout layer and implemented early stopping on a held-out validation set to prevent overfitting.

2.8.4.4 Other Baseline Monitors

To assess the effectiveness of scenario-specific adversarial training, we developed three context-aware baseline monitors, each implementing the same SCS STL logic used in the proposed context-aware monitors, but (1) without refining the thresholds (referred to as the CAWOT monitor), (2) with the thresholds learned from the fault-free data set, and (3) with the thresholds learned from all the populations' faulty data.

2.8.5 Metrics

We introduce the following metrics for the evaluation of system resilience and performance of safety monitors:

- ***Hazard Coverage*** is defined as the conditional probability that given activation of a safety-critical fault in the system by FI, it leads to an unsafe system state or a hazard.
- ***Time-to-Hazard (TTH)*** measures the time between activation of a fault (t_f) to occurrence of a hazard (t_h) (Fig. 2.8).
- ***Prediction Accuracy*** represents the performance of the safety monitors in accurate prediction of hazards, measured using false positive rate (FPR), false negative rate (FNR), accuracy (ACC), and F1 score. Using the traditional point-wise binary classification metrics, an FP is declared for all the samples in a simulation where the monitor detects a hazard and the ground truth indicates no hazard. But for hazard *prediction*, it is desirable that a monitor generates alerts *before* a hazard happens. So we adopt a modified version of standard classification metrics [106], proposed for sequential data [1] [107] [108], where a tolerance window before the start time of hazard (t_h) is used for calculation of the metrics (see Fig. 2.8). Table 2.4 shows the confusion matrix with a tolerance window.
- ***Reaction Time*** is the time difference between a monitor alert (t_d) and the occurrence of a hazard (t_h) (Fig. 2.8). This is the maximum time we have for taking any mitigation

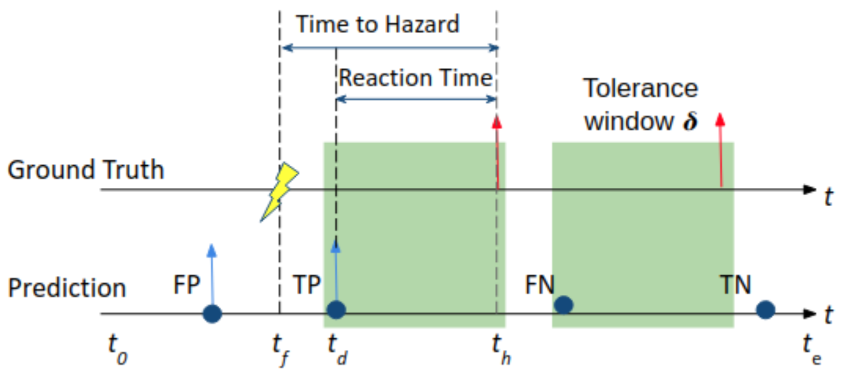


Figure 2.8: Hazard prediction accuracy with tolerance window δ (green area): TP: Hazard (red arrow) occurs no latter than δ after a prediction (blue arrow); FP: No hazard happens in [0,δ] after an alert; FN: Hazard occurs without a prediction in the window δ ahead; TN: No hazard happens in [0,δ] after a negative prediction.

Table 2.4: Confusion matrix for sequential data with tolerance window δ, modified from [1].

	Ground Truth Positive	Ground Truth Negative
Predicted Positive	$\sum_{t'=t-\delta'_t}^{t} P(t') > 0 \&\& \sum_{t'=t}^{t+\delta} G(t') > 0$	P(t)>0 && $\sum_{t'=t}^{t+\delta} G(t') == 0$
Predicted Negative	$\sum_{t'=t-\delta'_t}^{t} P(t') == 0 \&\& \sum_{t'=t}^{t+\delta} G(t') > 0$	P(t)==0 && $\sum_{t'=t}^{t+\delta} G(t') == 0$

* P(t)/G(t): Prediction/Ground truth at time t; $t - \delta'_t$: Start time of a window δ, ending with a positive ground truth, that includes t.

action before the hazard happens, with positive values representing *early detection*, and measures the timeliness of the monitor.

- ***Recovery Rate*** is the percentage of potential hazards that are prevented by the safety monitor's mitigation strategy and is affected by both the prediction accuracy and latency.
- ***Average Risk*** is a metric for assessing the impact of monitor performance on patient safety by considering the *consequences* of both FP and FN cases and the possibility of harm to patient. FNs put the patient in a hazardous situation without any warning or mitigation, and FPs not only bother the patient with unnecessary alerts but might also cause new hazards after needless mitigation. It is defined as follows:

$$Risk_{avg} = \frac{1}{N}[\sum{}_{i=1}^{N_{FN}} \bar{RI}(i) + \sum{}_{i=1}^{N'_P} \bar{RI}(i)] \tag{2.13}$$

where, $\bar{RI}(i)$ is the average risk index (for APS, defined as BG Risk Index in Section 2.7.2) of *ith* simulation, N is the total number of simulations, N_{FN} is the number of

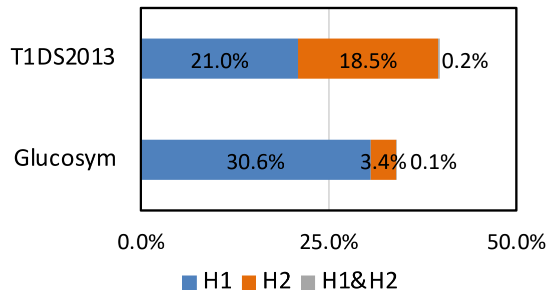
FN cases, and N'_P is the number of new hazards that are introduced by mitigation of FP cases.

2.8.6 Results

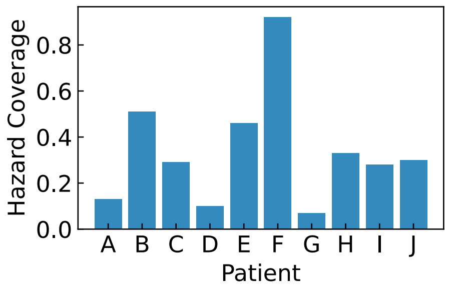
2.8.6.1 Resilience of Baseline Systems without Monitors

We first analyzed the resilience of the baseline OpenPilot and OpenAPS control software, which are already designed with safety features (e.g., forward collision warning [104] or a maximum threshold and an auto-adjusted control algorithm [109]), in the presence of faults without any safety monitors.

Effectiveness of FI: The experimental results demonstrated that our Fault Injection (FI) approach achieved an overall hazard coverage of 33.9% on the Glucosym simulator, 39.3% on the T1DS2013 simulator, and 39.9% on the OpenPilot platform. These results underscore both the effectiveness of our FI engine in generating faulty data for adversarial training and the vulnerabilities of the control software in handling safety-critical faults and attacks. Fig. 2.9 illustrates that FI covered all hazard types, with a predominant occurrence of hazard type H1 in Glucosym simulator experiments, thereby increasing the risk of hypoglycemia. Moreover, the hazard coverage varied significantly across different patient profiles, ranging from 6.7% to 92.4% across ten patients. This variability suggests the importance of specifying patient-specific and context-dependent safety requirements



(a)



(b)

Figure 2.9: (a) Hazard coverage of each hazard type for APS; (b) Hazard coverage of each patient.

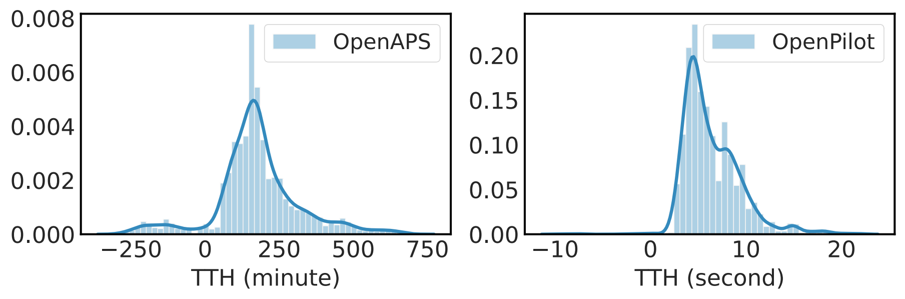


Figure 2.10: Time to hazard (TTH) distribution.

when designing monitors.

System Resilience: We assessed the resilience of OpenPilot and OpenAPS using the Time-to-Hazard (TTH) metric, aiding in the specification of time requirements for hazard prediction and mitigation.

Fig. 2.10 illustrates an average TTH of approximately 3 hours across all simulation data from OpenAPS. Notably, the human body exhibits significant latency and operates as a slow dynamic system, necessitating several hours for blood glucose (BG) levels to equilibrate and for insulin to elicit an effect. Additionally, 7.1% of hazardous simulations reported a TTH less than zero, indicating hazards occurring prior to any faults being injected into the controller, highlighting potential shortcomings in APS control algorithm.

In contrast, OpenPilot functions as a considerably faster control system, boasting an average TTH of 6.4 seconds. Consequently, different control action sequence lengths must be considered for safety monitoring purposes.

2.8.6.2 Monitor Prediction Accuracy

Context-Aware Monitors vs. Non-ML Monitors: Table 2.5 provides the average performance of the CAWT monitor across all the patients/fault scenarios in comparison to the non-ML-based baseline monitors, Guidelines and MPC.

In the assessment of APS, the CAWT monitor demonstrated superior performance across both the Glucosym and T1DS2013 simulators. While the Guideline monitor exhibited a slightly lower False Negative Rate (FNR) than the CAWT monitor in the T1DS2013

Table 2.5: Performance of CAWT monitor vs. non-ML monitors.

Simulator	**Monitor**	**No. Sim.**	**Hazard%**	**FPR**	**FNR**	**ACC**	**F1 Score**
Glucosym	Guideline	8820	33.9%	0.02	0.32	0.95	0.72
	MPC	8820	33.9%	0.02	0.34	0.95	0.71
	CAWOT	8820	33.9%	**0.01**	0.30	0.96	0.81
	CAWT	8820	33.9%	**0.01**	**0.02**	**0.99**	**0.96**
	MLP	8820	33.9%	0.02	0.07	0.97	0.89
	LSTM	8820	33.9%	0.04	0.06	0.96	0.81
T1DS2013	Guideline	8820	39.3%	0.07	$<$**0.01**	0.93	0.75
	MPC	8820	39.3%	$<$**0.01**	0.02	**1.00**	0.96
	CAWOT	8820	39.3%	0.02	0.04	0.98	0.89
	CAWT	8820	39.3%	$<$**0.01**	0.03	**1.00**	**0.97**
	MLP	8820	39.3%	$<$0.01	0.56	0.94	0.71
	LSTM	8820	39.3%	$<$0.01	0.06	0.99	0.95
OpenPilot	MPC	4800	39.9%	0.01	0.90	0.79	0.17
	CAWOT	4800	39.9%	0.29	0.12	0.76	0.66
	CAWT	4800	39.9%	$<$**0.01**	0.05	**0.99**	**0.97**
	MLP	4800	39.9%	0.01	0.11	0.97	0.93
	LSTM	4800	39.9%	0.01	$<$**0.01**	**1.0**	**0.99**

simulator, it produced more false alarms and yielded a 22.7% lower F1 score.

Regarding ADS, the MPC monitor failed to detect hazards for 90% of the instances, underscoring the vulnerability of the integrated FCW safety mechanism to attacks. Conversely, the CAWT monitor consistently delivered reliable performance in accurately forecasting hazards, showcasing up to a 4.7-fold enhancement in the average F1 score.

Context-Aware Monitors vs. Baseline ML Monitors: In the case of ADS, the LSTM monitor demonstrated superior performance compared to other baselines. However, the CAWT monitor showcased an F1 score comparable to the LSTM monitor while employing a simpler and more transparent model. Moreover, by adjusting the length of the control action sequence considered at each time step, the CAWT monitor has the potential to achieve even higher F1 scores and accuracy than the LSTM monitor, a topic we will delve into further in Section 2.9.

In APS case studies, the CAWT monitor consistently outperformed other baseline ML monitors in both the Glucosym and T1DS simulators, exhibiting an improvement in F1 score ranging from 7.9% to 36.6% while maintaining low FNR and FPR.

Table 2.6: Performance of context-aware monitor using thresholds learned from different data traces in APS.

Threshold	**FPR**	**FNR**	**ACC**	**F1 Score**	**EDR**
Fault-free	0.01	0.27	0.96	0.83	95.1%
Faulty	0.01	**0.02**	**0.99**	**0.96**	**99.2%**
Population-based	0.08	0.08	0.92	0.89	92.2%
Patient-specific	**0.01**	**0.00**	**0.99**	**0.96**	**100.0%**

Context-Aware Monitors vs. Other Baselines: We conducted further evaluations of the CAWT monitor's performance under various conditions, including without refining thresholds (CAWOT), with thresholds learned from all patients' data traces with and without fault injection, patient-specific thresholds learned from each patient's faulty data traces, and population-based thresholds learned from all patients' erroneous data. For the population-based model, we trained the thresholds on data from seventy percent of randomly selected patients and tested the model on the remaining thirty percent of patients.

Table 2.5 illustrates that without refining the thresholds of SCS rules, the CAWOT monitor experienced a decrease in F1 score ranging from 8.2% to 32.0%, underscoring the significance of optimizing safety requirements. However, it still outperformed the MPC and Guideline monitors in the Glucosym simulator, underscoring the advantages of context awareness.

Table 2.6 highlights the performance of the context-aware monitor using thresholds learned from fault-free data, indicating that it detected unsafe control actions before hazards occurred in 95.1% of true-positive cases but failed to generate alerts for hazardous situations in 27% of simulations. Through adversarial training and refinement of SCS formulas with faulty data, the monitor's performance improved by 4.1% in early detection rate (EDR) and 15.7% in F1 score.

Additionally, the context-aware monitor with patient-specific thresholds demonstrated superiority over a population-based monitor, achieving a 7.6% increase in accuracy and

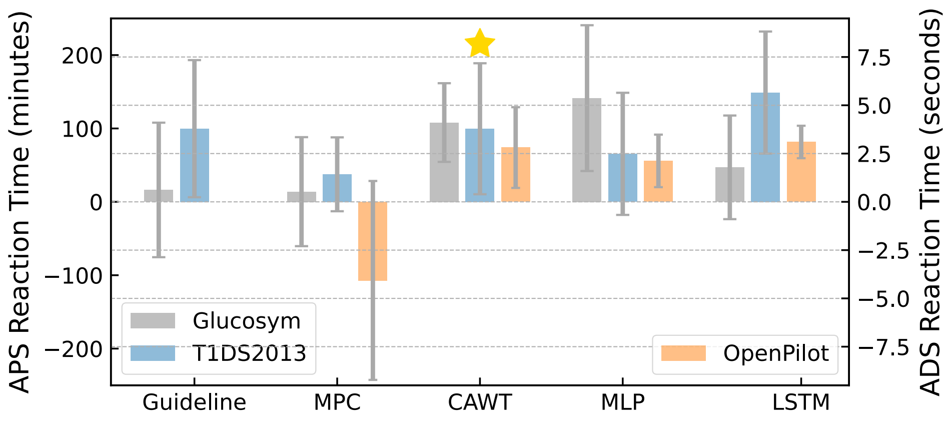


Figure 2.11: Average reaction time for each monitor.

a 7.8% increase in EDR. Furthermore, the patient-specific context-aware monitor maintained low FPR and FNR, resulting in a 7.9% higher F1 score.

2.8.6.3 Monitor Timeliness

Fig. 2.11 illustrates the reaction time (as defined in Section 2.8.5 and Fig. 2.8) for each monitor. The CAWOT monitor is not included here due to its inferior performance compared to the CAWT monitor (refer to Table 2.5).

Across all simulators, the CAWT monitor consistently demonstrated effective performance in ensuring a safe reaction time. In the case of APS, the average reaction time was approximately 100 minutes, exceeding the peak activity of insulin, typically between 60 and 90 minutes [110]. For ADS, the average reaction time aligned with the safe headway time of 2 to 3 seconds [71].

Conversely, the non-ML baseline monitors exhibited the poorest performance in timely detection of unsafe control actions. This can be attributed to their fixed threshold designs, which limited their adaptability across different patients and scenarios. Moreover, the MPC monitor could only predict hazards within a short window ahead of time in the Glucosym and T1DS2013 simulators and demonstrated a negative average reaction time in OpenPilot, indicating delayed detection and an inability to prevent potential hazards.

Benefiting from a wealth of collected data and scenario-specific models, the baseline

ML monitors outperformed the Guideline and MPC monitors. However, the performance of these baseline ML monitors varied considerably, with large standard deviations, and was not as consistent as that of the CAWT monitor.

2.8.6.4 Hazard Mitigation

We evaluated the mitigation performance of the CAWT monitor against two baseline monitors: the MLP monitor, which exhibits a comparable F1 score to the LSTM but employs simpler logic, and the MPC monitor, selected as the top-performing non-ML-based baseline monitor. We reran the simulations with each monitor and the mitigation algorithm (Algorithm 1).

Table 2.7: Mitigation performance of the CAWT monitor and three baseline monitors using the same mitigation strategy.

Monitor	**CAWT**	**MLP**	**MPC**
Recovery Rate	**54.0%**	39.0%	4.3%
No. New Hazard	**8**	177	123
Avg. Risk	**0.02**	0.68	0.22

Table 2.7 illustrates that the CAWT monitor successfully averted 54% of previously occurring hazards, while introducing only eight new hazards due to false alarms, resulting in the lowest average risk among the monitors. In contrast, the MPC baseline monitor, with the same mitigation algorithm, achieved a recovery rate of only 4.3%, highlighting the drawbacks of inadequate reaction time. Despite achieving sufficient long average reaction time, the MLP monitor prevented only 40.3% of hazards and introduced the largest number of new hazards, underscoring the importance of being context-aware. These findings underscore the significance of having a sufficiently prompt reaction time in ensuring a better recovery rate, while also emphasizing the benefits of context-awareness in enhancing overall mitigation performance.

2.8.6.5 Adaptive Adversaries

A significant challenge in anomaly detection is the presence of adaptive adversaries who leverage knowledge of existing safety mechanisms to adjust attack parameters, evade detection, and trigger adverse events [99, 100, 111]. To assess the effectiveness of our proposed safety monitoring approach against such stealthy attacks, we consider a highly sophisticated attacker equipped with knowledge of (1) the logic of our safety monitor, (2) the parameters (e.g., β_i in Table 2.1), and (3) the format of control commands.

In our implementation, to evade detection, the stealthy attacks are only launched when the target monitor is not triggered to check the possibility of unsafe control actions. More specifically, the malicious changes to the controller state variables are still injected at random start times, but only remain active when none of the safety context conditions specified for the monitor (in the third column of Table 2.1) are held true. For example, to cause a forward collision with the lead vehicle, the attacker can keep accelerating the Ego vehicle until *Headway Time* (HWT) reaches an unsafe threshold. To avoid being detected by the context-aware monitor, the Ego vehicle is required to decelerate until HWT goes back to a safe range (e.g., β_{21}-β_{26} in Table 2.1) or until the ego vehicle is slower than the lead vehicle (e.g., RS<0), which will not trigger the context condition for the monitor.

Our experiments encompass three categories of stealthy attacks [22], introduced according to the scenarios outlined in Table 2.2, as follows: (1) surge attacks aiming to maximize the attack impact swiftly by maximizing the attack value (scenario *Max*), (2) bias attacks striving to extend the duration of the attack while remaining undetected by minimizing the attack value (scenario *Min*), and (3) random attacks randomly selecting attack values (scenario *Add/Sub*). These scenarios were tested with 8820, 8820, and 4800 simulations in the Glucosym, T1DS, and OpenPilot simulators, respectively. The experimental outcomes reveal that the stealthy attacks directed at the context-aware monitors with refined thresholds (CAWT) do not result in any hazards due to their failure to trigger any of the safety context conditions.

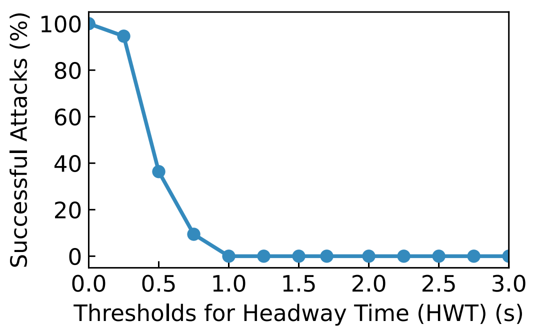


Figure 2.12: Stealthy attack success rate for different settings of the thresholds (β_{21}-β_{26} in Table 2.1) for the monitor parameter *Headway Time (HWT)*.

However, the efficacy of the CAWT monitor against adaptive adversaries and the success of stealthy attacks in causing hazards depend on the completeness and accuracy of the generated SCS and the monitor parameters, as further discussed in Section 2.10. Using ADS as an example, Fig. 2.12 illustrates the impact of thresholds utilized for the CAWT monitor on the success rate of stealthy attacks. Specifically, it depicts the percentage of simulations across different types of stealthy attacks wherein a hazard occurred while remaining undetected by the monitor for varying values of the safety thresholds (β_{21}-β_{26} in Table 2.1) for the parameter HWT. The success rate of the attack decreases with the increase of HWT thresholds, with no hazards occurring anymore when the thresholds surpass 1 second (the thresholds learned for HWT to develop CAWT monitors typically fall between 2-3 seconds). A similar effect may be observed if the attacker can identify any additional safety context conditions under which hazards may occur (additional rows in Table 2.1) but were not considered in the design of the target monitor.

2.8.6.6 Evaluation on a Clinical Trial Dataset

To further assess the effectiveness of the proposed safety monitor in real-world scenarios using realistic data, we conducted performance testing using a publicly available diabetic dataset known as DCLP3 [112]. This dataset originates from a clinical trial involving 168 diabetic patients aged 14 to 71 years old who underwent six-month treatment with the only FDA-approved closed-loop APS, t:slim X2 with Control-IQ Technology.

We partitioned each patient's data into 180-day segments and analyzed 150 iterations of glucose readings and insulin records per day, resulting in a total of 30,240 days of data and 4,536,000 samples (30,240 * 150). Hazard events in the dataset were labeled using the risk index approach outlined in Section 2.7.2.

The experimental results revealed that the context-aware monitors developed using the proposed method achieved an F1 score of 0.86. Furthermore, these monitors accurately predicted actual adverse events observed in the data, encompassing both CGM-measured hyperglycemic events (defined as a period of at least 15 consecutive minutes with $BG < 54\ mg/dL$) and CGM-measured hypoglycemic events (defined as a period of at least 120 consecutive minutes with $BG > 300\ mg/dL$) [112], with a success rate of 99.5%.

2.8.6.7 Resource Utilization

We ran the simulations with different safety monitors and without a monitor a thousand times and calculated the average time overhead for each safety monitor. Results showed that the CAWT monitor has the lowest average time overhead of 252.7 us among all the safety monitors, while the time overhead of MPC, Guideline, MLP, and LSTM monitors was 123.9 ms, 664.1 us, 30.7 ms, and 32.6 ms, respectively.

2.9 Discussion

Our experiments provided the following key insights:

Both the OpenPilot and OpenAPS control software cannot tolerate safety-critical faults. Despite OpenPilot being an advanced control system widely utilized in real-world road driving and equipped with an integrated forward collision warning function, and OpenAPS being a fully automated system with built-in safety features, they proved unable to withstand the simulated attacks and faults. In 13.8% of the APS simu-

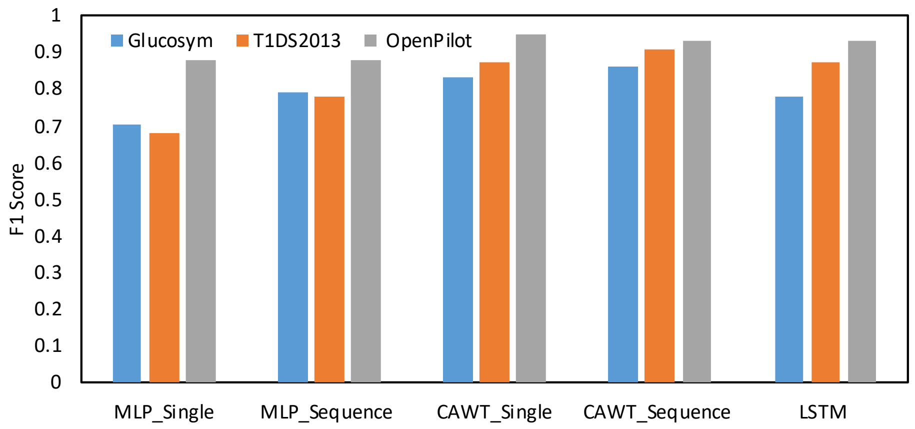


Figure 2.13: Performance of each monitor based on a single control action or a sequence of control actions.

lations, patients were predicted to experience severe hypoglycemia. Additionally, in 81% of the simulated driving scenarios, OpenPilot failed to issue alerts before a collision took place. Furthermore, certain hazards occurred even in the absence of any fault injections.

Adversarial training using scenario-specific data improves the performance of CAWT monitors. As shown in Section 2.8.6.2 the CAWT monitor with thresholds learned from faulty data of a specific patient outperformed other context-aware monitors that were designed with the exact SCS logic but with different thresholds. These results reconfirm that adversarial training and refinement of SCS formulas using the faulty data is important in improving the CAWT monitor's performance. Furthermore, each patient has different biomedical characteristics and tolerance levels to the injected insulin amounts. Thus, the safety monitor logic needs to be refined for each patient or scenario.

By considering the sequence of control actions, the CAWT monitors can generate more accurate alerts. Fig. 2.13 illustrates the performance of the CAWT and MLP monitors, considering either a sequence of control actions or a single control action, in comparison to the LSTM monitor. Both the MLP and CAWT monitors, which take into account a sequence of control actions, achieved a lower False Positive Rate (FPR) and a higher F1 score, with a slightly higher False Negative Rate (FNR), than the same monitors that used only a single control action for both the Glucosym and

T1DS2013 simulators. Additionally, after considering a sequence of control actions, the MLP monitor demonstrated slightly better performance than the LSTM monitor in the Glucosym simulator and reduced the performance gap with the LSTM monitor in the T1DS2013 simulator. However, for ADS, which operates at a much faster pace than APS, the CAWT monitor based on a single control action achieved superior performance, indicating that the length of control action sequences is an important consideration in designing safety monitors for different CPS.

Weakly supervised context-aware monitor outperforms ML-based monitors. Our experiments showed that in most situations the CAWT monitor could achieve a better or comparable performance to the ML-based monitors that we explored in this work. There are several other advantages that a CAWT monitor has over ML-based monitors:

(1) *Data Limitation and Corner Cases.* Fully supervised ML-based monitors tend to suffer from overfitting to the datasets they have been trained on [113]. For example, we evaluated their performance on datasets collected from fault-free simulations, and results showed at least a 48.9% drop in F1 score compared to their performance on faulty data. In comparison, the F1 score of the CAWT monitor only decreased 3.9% because it was trained using a weakly supervised approach that only uses faulty data to tighten the SCS thresholds.

(2) *Application Strategies and Resource Limitations.* Implementing the CAWT monitor in real-world applications involves collecting data from simulation or real-time operation over a specific period and refining unknown thresholds for each SCS rule offline. At runtime, the CAWT monitor loads the learned thresholds and functions as a wrapper integrated with the CPS controller, requiring minimal resources. In contrast, ML-based monitors must load pre-trained models and utilize significantly more resources than the CAWT monitor.

(3) *Monitor Safety and Interpretability.* Neural network classifiers are often considered

black-box systems [114], lacking transparency and explainability in their decisions. They are also susceptible to adversarial attacks, perturbations, and input noise [115], leading to misclassification. Conversely, our proposed CAWT monitor relies on a weakly supervised and transparent model, which is simpler to verify, update, and protect.

2.10 Threats to Validity

Sensor Perturbations: This work primarily focuses on faults and attacks targeting the control software of safety-critical CPS. Any perturbations in sensor data could potentially affect both the controller and the safety monitor's behavior. However, several existing methods in the literature [22, 32, 95, 116] can be integrated with our safety monitor to protect sensor data and actuator commands observed by the monitor. Furthermore, slight disturbances in sensor data caused by environmental noise can be mitigated by the typical robustness features of the control system (e.g., the use of a PID algorithm) to ensure control actions are not compromised. Additionally, combining domain knowledge with data could enhance the monitor's robustness against small accidental or malicious perturbations in the input data [117].

SCS Completeness: The performance of the proposed monitor heavily relies on the accuracy and completeness of the generated SCS, which may be challenging to derive for highly complex systems. However, our method only utilizes a subset of state variables that can adequately represent the system's physical states and dynamics. Furthermore, inaccurate or incomplete specification is a common issue in the design and verification of any controller/monitor. We aim to mitigate such manual errors by proposing a formal framework for designers, in collaboration with domain experts, to generate SCS.

Sim-to-Real Gap: Simulations are crucial for advancing research rapidly while minimizing risks associated with testing in real operating environments and potentially harming real individuals such as patients, drivers, and pedestrians. Nonetheless, disparities between simulations and real-world implementations can undermine the validity of proposed methods. For example, in APS simulations, we only model a patient going to sleep after dinner without considering other physical activities or multiple meal consumption scenarios. To mitigate the sim-to-real gap, we utilize real control software (deployed with actual diabetic patients [118,119] or in real vehicles [83]) and patient simulators approved by the FDA for clinical testing or based on actual patient profiles [26,102]. Additionally, we incorporate realistic high-risk driving scenarios defined by NHTSA for ADS evaluation. Furthermore, validation using publicly available datasets from clinical trials [112] confirms the effectiveness and validity of the proposed approach [3].

2.11 Related Work

Anomaly Detection in CPS. Previous research efforts in anomaly detection have primarily focused on identifying safety-critical attacks that target sensor data [22,31,46,120]. However, there has been relatively less emphasis on detecting attacks that compromise the functionality of controllers by directly manipulating their internal logic and variables [48]. Furthermore, existing methods often rely on simplistic linear models of physical systems, which may not fully capture their dynamics [25, 41], or black-box ML models [32, 55], which can suffer from a lack of generalization and transparency.

Our work differs from these prior approaches by focusing on detecting faults and attacks that affect controller internal variables and outputs. Additionally, we introduce the concept of preemptive detection of early signs of hazards, rather than detecting them after they have occurred. This proactive approach enables timely mitigation and recovery, which may be crucial for preventing adverse consequences.

Run-time Monitoring with STL Learning: Several recent works [62, 121, 122] have focused on approaches for monitoring, learning, and controlling CPS behaviors using STL. For instance, [123] applied STL learning and monitoring for anomaly detection in CPS, while [124] utilized STL learning to characterize behaviors of Type 1 Diabetes

(T1D) patients. However, our work distinguishes itself from these prior efforts by introducing a formal framework that combines STL formalism for specifying safety contexts with scenario-specific STL learning. This approach enables the design of context-aware monitors capable of predicting and mitigating safety hazards effectively.

2.12 Conclusion

In this chapter, we introduced a formal framework for the control-theoretic specification and refinement of the safety context, which can be used for designing context-aware safety monitors capable of predicting and mitigating hazards in CPS as well as for design-time safety validation. We conducted case studies using closed-loop APS and ADS simulation systems to evaluate the effectiveness of the proposed method. Experimental results demonstrated that our monitor outperforms several baseline monitors developed using medical guidelines, MPC, and ML techniques in accurately and timely predicting hazards. Additionally, our monitor exhibited stable performance in ensuring sufficient reaction time and mitigating hazards effectively.

Chapter 3

Combined Knowledge and Data Driven Hazard Prediction and Mitigation

3.1 Overview

The existing literature on CPS safety and security predominantly emphasizes anomaly detection or run-time monitoring [20, 46–48], with comparatively limited focus on hazard mitigation or recovery strategies. Previous approaches to maintaining CPS safety after anomaly detection have relied on redundant hardware or software sensors or components [31, 125]. However, such methods often increase system complexity and may still be vulnerable to common vulnerabilities shared across replicas. Efforts have also been made to respond to anticipated hazards by triggering fail-safe modes (e.g., freezing insulin pumps in APS) [126, 127] or manual remediation (e.g., generating alerts, handing over control to human operators) [111, 128]. However, these approaches either cannot maintain regular system operation or face challenges in ensuring the safety of manual recovery due to tight timing constraints and short reaction times.

Recent research primarily focuses on designing runtime monitors [47, 59] and simplex controllers [130] that serve as backups when the primary controller is compromised. Model-based approaches in this domain involve developing linear or non-linear models of system dynamics and their interactions with the environment to detect anomalies [22]

This chapter contains material from the previously published works [53, 117, 129], coauthored with H. Alemzadeh, J. Aylor, B. Ahmed, P. Asare, M. Kouzel, and C. Smith, copyrighted by IEEE.

and generate recovery actions [23, 24]. However, developing models capable of fully capturing complex system dynamics and unpredictable human physiology and behavior (e.g., glucose sensitivity to insulin in APS) poses a challenge. Additionally, the accumulation of approximation errors between actual system states and model predictions with each operational cycle limits the effectiveness of model-based recovery approaches over time.

Data-driven approaches leveraging machine learning (ML) have demonstrated enhanced accuracy and success rates in attack detection and recovery [31, 33, 55]. However, they frequently rely on black-box deep learning models, which suffer from transparency and robustness issues [117]. Moreover, these methods often initiate recovery actions *after* attacks have caused noticeable deviations in system states or resulted in hazards [20, 46], potentially rendering them ineffective in preventing adverse events.

To address the gap in hazard mitigation in CPS and overcome the limitations of solely model-based and data-driven solutions, this work proposes a combined knowledge and data-driven approach, termed *KnowSafe*, for predicting and mitigating safety hazards caused by faults or attacks in real-time. *KnowSafe* integrates expert "Knowledge" of safety context specification (introduced in Section 2.3) or domain-specific "Safety constraints" with data from the closed-loop CPS operation to develop a safety engine. This engine can be incorporated into a CPS controller's interface to infer system context, anticipate impending hazards, and avert the execution of unsafe control actions by generating preemptive and corrective control actions.

3.2 Design Challenges

Anomaly detection is a critical step in hazard mitigation and recovery. Previous works on anomaly detection and recovery mainly focus on comparing the predicted states with actual states based on sensor data and waiting until a large deviation over a detection window to raise an alarm [31, 46]. However, relying on predefined thresholds for detect-

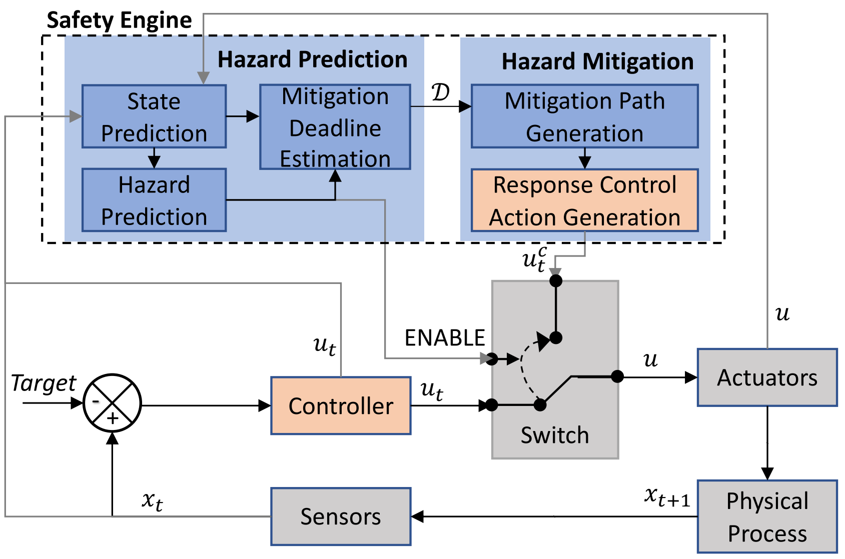


Figure 3.1: Combined knowledge and data driven safety engine for hazard prediction and mitigation.

ing deviations may lead to delayed detection and high false positive rates, resulting in mitigation failures. Therefore, designing a new method that can detect or predict hazard occurrence in a timely and accurate manner is important yet challenging.

Further, efficient hazard mitigation encounters additional challenges in generating optimal mitigation actions that can return the system back to the safe region as quickly and smoothly as possible while ensuring the satisfaction of safety requirements throughout the mitigation process.

3.3 Approach Design

To address these challenges, this work proposes a combined knowledge and data-driven approach for designing a safety engine capable of predicting and mitigating potential hazards at runtime. Fig. 3.1 illustrates the overall design of the safety engine, comprising a hazard prediction module (discussed in Section 3.3.2) and a hazard mitigation module, which includes mitigation path planning (covered in Section 3.3.3) and corrective control action generation (explained in Section 3.3.4).

During each control cycle, the hazard prediction module receives a sequence of system states or their transformations, inferred from sensor data, as input and accurately predicts the probability of hazard occurrence using a customized classification or regression model. The duration between the current time and when the predicted hazard is expected to occur is defined as the *mitigation deadline* or the timeframe within which corrective or responsive control actions must be initiated to mitigate hazards. If potential hazards are identified, the hazard mitigation module devises an optimal path and generates a corresponding sequence of control actions to guide the system away from unsafe conditions and return it to a safe state, all while adhering to the mitigation deadline and other domain-specific safety constraints.

3.3.1 ML Optimization with Customized Loss Functions

To address the challenge of timely and accurate hazard prediction, we already introduced a weakly supervised learning method with STL parameter optimization in Section 2.5. However, the effectiveness of this approach heavily relies on the completeness of the generated SCS. In this chapter, we present a new approach that combines the generated SCS with ML techniques to improve hazard prediction accuracy and timeliness while not requiring the complete specification of SCS rules.

Problem Statement: Specifically, we model the task of detecting an unsafe control action as a context-specific conditional event, as shown below:

$$y_t = p(\exists t' \in [t, t+T] : x_{t'} \in \mathcal{X}_h | f(X_t), f(U_t)) \tag{3.1}$$

For a given control action sequence U_t executed under the system state sequence X_t, the ML model produces a binary output y_t that categorizes U_t as safe or unsafe.

Data and Knowledge Integration: We encode the STL formulas generated for SCS as a custom loss function [131] that penalizes the ML model during the training process if the prediction does not align with the specified safety properties:

$$loss = \eta loss_{ex} + (1-\eta)\mathcal{L}(y_t, I\left(\bigvee_{\Phi_h \in SCS} f(\mu(X_t)) \models \Phi_h\right)) \tag{3.2}$$

where, $loss_{ex}$ represents the existing or original loss function of the ML model, such as cross-entropy loss. The parameter w denotes a weight parameter, while y_t stands for the output prediction of the ML model. The function $I(\cdot)$ serves as an indicator function that determines whether the aggregated values of the estimated state variables for a measurement window, denoted as $f(\mu(X_t))$, satisfy any of the unsafe specifications in the STL formulas Φ_h (with unrefined thresholds). The specific value assigned to the weight parameter η depends on the design requirements and system scenarios. A smaller weight parameter implies a greater influence of the system context and safety specifications on the training process. In this study, we select a value for the weight parameter to ensure that the additional loss is comparable to the original loss of the ML model's output layer.

3.3.2 Reachability Analysis and Hazard Prediction

In scenarios where specifying the Safety Context Specifications (SCS) proves challenging, particularly in considerably complex systems, we additionally introduce a general hazard prediction approach. This method estimates the potential sequence of future system states and determines whether they are likely to fall within any unsafe operational regions.

Problem Statement: We specifically define the following binary classification problem:

$$y_t = \begin{cases} 1, & \text{if } \exists t' \in [t, t+N] : \{\hat{x}_{t'} \in g(X_t, U_t)\} \subset \mathcal{X}_h \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

where, the function $g(\cdot)$ represents a prediction model that, given an input state sequence $X_t = \{x_{t-k}, ..., x_t\}$ and the control action sequence $U_t = \{u_{t-k}, ..., u_t\}$, predicts the system state trajectory $\hat{X}_{t+N} = \{\hat{x}_{t+1}, ..., \hat{x}_{t+N}\}$ within a prediction window of N control cycles. A hazard is anticipated if any predicted state $\hat{x}_{t+i}$ lies within the unsafe region $\mathcal{X}_h$. Additionally, the mitigation deadline, $\mathcal{D}$, is estimated by determining the difference between the current time t and the minimum t' within the prediction window such that the system state $\hat{x}_{t'}$ is unsafe.

$$\mathcal{D} = min\{t' \in [t, t+N]\} - t : \hat{x}_{t'} \in \mathcal{X}_h \tag{3.4}$$

Domain Knowledge Specification: Ensuring the accuracy and realism of predicted system state sequences involves verifying a set of application-specific constraints. These constraints may involve checking whether the changes from the initial state to the predicted states fall within reasonable bounds. Such properties are often described in human-interpretable formats over a transformed state space, which could represent derivatives, polynomial combinations, or other functions of state variables (e.g., the first derivative of blood glucose to represent the rate of change).

Projecting both the current state x_{t+i} and its predicted state in the next step $\hat{x}_{t+i+1}$ onto the transformed state space $\mu(x) = (\mu_1(x), \mu_2(x), ..., \mu_m(x))$, we define the δ-reachable state of $\mu(x_t)$ as a set or region $\mathcal{S}(\mu(x_t), \delta)$. This region $\mathcal{S}$ encompasses states whose differences from $\mu(x_t)$ are constrained by the parameter set δ. This concept is formalized as follows:

$$\mathcal{S}(\mu(x_t), \delta) := \{\forall s_t \in \mathcal{S}, ||s_t - \mu(x_t)|| \leq \delta\} \tag{3.5}$$

where $||\cdot||$ denotes the norm used to measure the distance between two states in the transformed space. The parameter set $\delta = (\delta_1, \delta_2, ..., \delta_m)$ comprises parameters corresponding to each variable in $\mu(x)$. These parameters can be determined based on domain-specific guidelines, design requirements, practical experience, or statistical analysis of past data. They define the acceptable range of variations for each transformed variable, ensuring that the predicted states remain within realistic bounds.

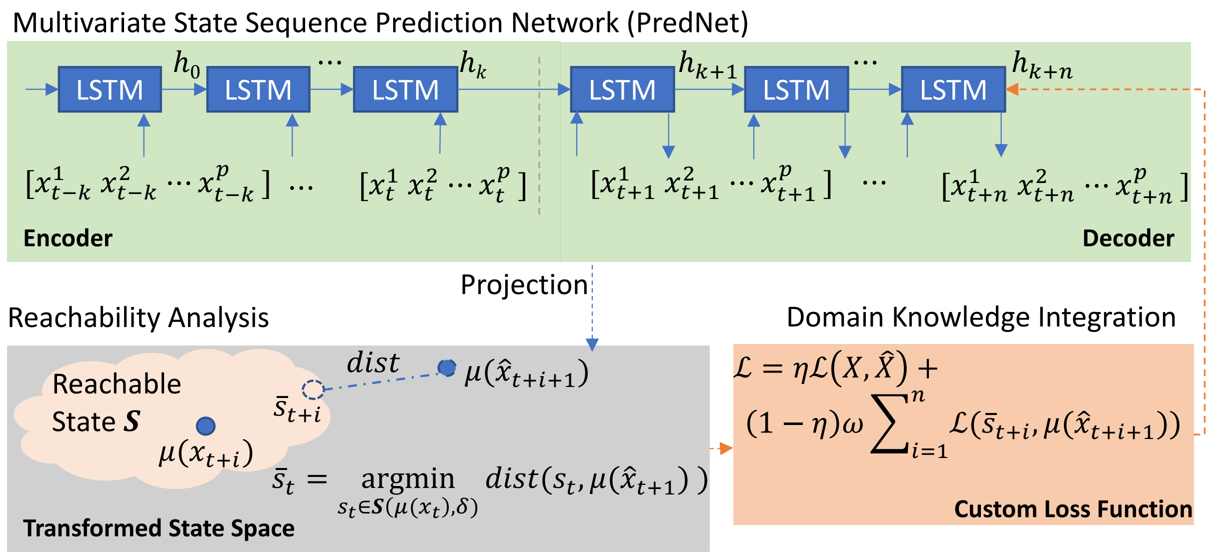

Multivariate State Sequence Prediction Network (PredNet)
LSTM
h_0
h_k
h_{k+1}
h_{k+n}
$[x_{t-k}^1\ x_{t-k}^2 \cdots x_{t-k}^p]$
$[x_t^1\ x_t^2 \cdots x_t^p]$
$[x_{t+1}^1\ x_{t+1}^2 \cdots x_{t+1}^p]$
$[x_{t+n}^1\ x_{t+n}^2 \cdots x_{t+n}^p]$
Encoder
Decoder
Projection
Reachability Analysis
Domain Knowledge Integration
Reachable State S
$\mu(x_{t+i})$
$\bar{s}_{t+i}$
dist
$\mu(\hat{x}_{t+i+1})$
$\bar{s}_t = \underset{s_t \in S(\mu(x_t),\delta)}{\operatorname{argmin}}\ dist(s_t, \mu(\hat{x}_{t+1}))$
Transformed State Space
$\mathcal{L} = \eta\mathcal{L}(X,\hat{X}) + (1-\eta)\omega\sum_{i=1}^{n}\mathcal{L}(\bar{s}_{t+i}, \mu(\hat{x}_{t+i+1}))$
Custom Loss Function

Figure 3.2: Multivariate regression neural net for sequential states prediction (PredNet) with domain knowledge integration.

Data and Knowledge Integration: In this study, we introduce a novel multivariate regression neural network model, denoted as *PredNet*, designed to forecast the sequence of system states using data obtained from either closed-loop CPS simulations or real system operations (e.g., from clinical trials, as discussed in Section 3.4.5.1). Illustrated in Fig. 3.2, our prediction network adopts an encoder-decoder architecture that anticipates the future system states $x_{[t+1:t+N]}$ for $N = n$ steps based on a sequence of $k + 1$ previous system states. The domain-specific properties are incorporated into *PredNet* by appending a customized loss function as a regularization term to the original loss function $\mathcal{L}(X, \hat{X})$ (e.g., mean squared error), as depicted below:

$$\mathcal{L} = \eta\mathcal{L}(X, \hat{X}) + (1-\eta)\omega\sum_{i=1}^{n}\mathcal{L}(\bar{s}_{t+i}, \mu(\hat{x}_{t+i+1})) \tag{3.6}$$

$$\bar{s}_t = \underset{s_t \in \mathcal{S}(\mu(x_t),\delta)}{\operatorname{argmin}}\ dist(s_t, \mu(\hat{x}_{t+1})) \tag{3.7}$$

where, $\bar{s}_{t+i}$ represents the state within the δ reachable state of $\mu(x_{t+i})$, which minimizes the distance (e.g., Euclidean distance) to the predicted state transformation $\mu(\hat{x}_{t+i+1})$. The parameter ω is a weight parameter, which is zero when $\mu(\hat{x}_{t+i+1})$ falls within the reachable state S, and one otherwise. Additionally, $\eta \in [0, 1]$ denotes the weight of the original loss contributing to the total loss. The value of η is manually selected through hyper-parameter tuning. The custom loss function ensures realistic trajectories by penalizing *PredNet* during training whenever a predicted state lies outside the reachable set of the current state.

Furthermore, we establish a two-level hazard prediction pipeline comprising (i) a long-term prediction model (*PredNet-l*) capable of predicting a distant-future state sequence, allowing sufficient time to anticipate and mitigate potential hazards, and (ii) a short-term prediction model (*PredNet-s*) boasting higher accuracy than the long-term model to prevent overlooking any hazards.

3.3.3 Optimal Mitigation Path Generation

The subsequent critical phase in hazard mitigation involves determining the appropriate response actions for the system to safely achieve its objectives, such as remaining within the controller's target region. While a straightforward approach to implementing mitigation involves compensating control actions based on predefined rules or medical guidelines (e.g., increasing insulin injection to prevent hyperglycemia in APS or adjusting gas and brake inputs to avoid collisions in an Autonomous Driving System (ADS)), such a generic mitigation strategy overlooks the current system context and its interactions with the environment. Consequently, it may result in either over- or under-mitigation, potentially jeopardizing system users' safety [47]. Designing a mitigation mechanism that ensures a high recovery rate while minimizing the introduction of new hazards poses a significant challenge.

Our objective is to acquire a sequence of control actions that effectively mitigate hazards and maintain the system's safety within permissible regions. A straightforward approach would involve inputting all potential actions into the previously introduced prediction model and selecting the optimal action resulting in a future system state sequence closest to the safe region. However, this method could be computationally intensive and time-consuming since it necessitates traversing the entire action space and may not converge to an optimal solution within the specified mitigation deadline.

Hence, instead of directly producing recovery control actions, we initially identify the desired sequence of recovery states using a rapid path planning algorithm. Subsequently, we employ a control algorithm, either ML-based or model-based, to generate a corresponding sequence of control actions aligned with the identified recovery states. This two-step approach facilitates more efficient and effective hazard mitigation while accommodating the system's operational constraints and resource limitations.

Problem Statement: We formulate the generation of the mitigation path as an optimization problem aiming to swiftly return the system to a safe state, ideally before the onset of hazards, while evading the unsafe region. Our objective is to minimize the time required to mitigate the hazard, subject to the following constraints:

$$\min\{TimetoMitigation\};\ \text{s.t.} \tag{3.8}$$

$$\Delta x_i/T < \alpha_i^1; \forall i \in [1,p] \tag{3.9}$$

$$\Delta(\Delta x_i)/T^2 < \alpha_i^2; \forall i \in [1,p] \tag{3.10}$$

$$\{x\} \cap \mathcal{X}_h = \varnothing \tag{3.11}$$

$$PathLength < \mathcal{D} \tag{3.12}$$

where, where x represents the system state, T denotes the duration of a control cycle, and α_i^j signifies an application-specific constraint. $\mathcal{D}$ stands for the maximum mitigation budget available to counteract hazards and also serves as the upper bound for the length of the mitigation path ($PathLength$), as estimated by the hazard prediction module (see Fig. 3.1).

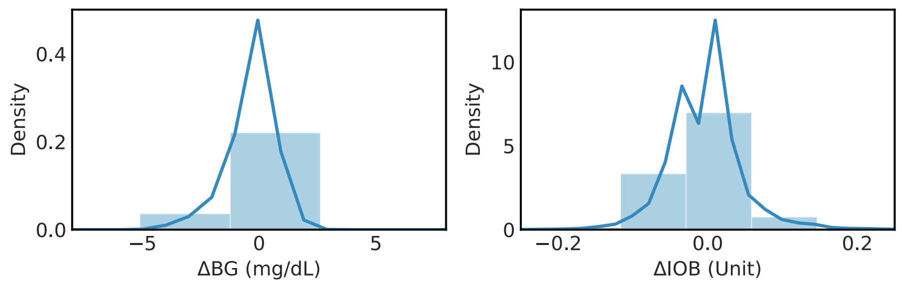


Figure 3.3: Example distribution of ΔBG and ΔIOB.

Domain Knowledge Specification: The constraints α_i^j are in place to ensure the fulfillment of safety properties and the smoothness of state transitions, and they are defined based on domain expertise or practical insights. For instance, in an APS scenario, a constraint could dictate that the change in blood glucose (BG) levels should not surpass $[-5, 3]$ mg/dL over a 5-minute interval [47]. Fig. 3.3 shows an example distribution of rate of BG changes in APS which is used to derive these constraints.

Data and Knowledge Integration: We introduce a mitigation path planning algorithm, named *SC-RRT**, designed to accommodate safety constraints. This algorithm is based on a modified version of the Rapidly-exploring Random Tree Star (RRT*) approach [132]. RRT* is known for its efficacy in handling problems involving obstacles and differential constraints, offering a wide search range, rapid search speed, and high computational efficiency, which makes it widely applicable in robotic motion planning. By leveraging *SC-RRT**, our methodology becomes potentially viable across a broader spectrum of CPS applications. Moreover, *SC-RRT** boasts lightweight implementation and greater transparency compared to neural networks, making it more suitable for deployment on devices with limited computational resources and memory capacity.

Algorithm 2 outlines the process of generating a mitigation path. At each iteration, the algorithm begins by randomly sampling a point in the state space (line 17) and establishing a new vertex in the random tree G toward the nearest vertex within the incremental distance Δx (lines 18-19). Importantly, domain knowledge is incorporated as safety constraints to eliminate invalid vertices during the random sampling step. Specifically, a

```
Algorithm 2: Mitigation Path Planing Algorithm
Input: Initial config $x_{init}$, Number of vertices $K$, Incremental distance $\Delta$x, Near Radius $\gamma$,
       Recovery Budget $D$
Output: RRT graph G, Optimal Goal Vertex $x_{optimal}$ , Minimum recovery path $path_{min}$
1  G.init($x_{init}$) {Init Optimal RRT}
2  $X_{dest} \leftarrow \emptyset$ {Vertices in Target Region}
3  Function UpdateCost($X_{near}$, $x_{new}$):
4      for $x_{near} \in X_{near}$ do
5          if ValidConnection($x_{near}, x_{new}$) then
6              if Cost($x_{near}$)+Dist($x_{nearest}, x_{new}$) $< cost_{min}$ then
7                  $x_{min} \leftarrow x_{near}, cost_{min} \leftarrow$
8                  Cost($x_{near}$)+Dist($x_{nearest}, x_{new}$)
9      end
10 return
11 Function RewriteTree($X_{near}$, $x_{new}$):
12     for $x_{near} \in X_{near}$ do
13         G.update($G, x_{near}, x_{new}$) {update shorter path with $x_{new}$}
14     end
15 return
16 for $k = 1$ to $K$ do
17     $x_{rand} \leftarrow$ RandomSampling()
18     $x_{nearest} \leftarrow$ FindNearestNode(G,$x_{rand}$)
19     $x_{new} \leftarrow$ NewConf($x_{nearest}$, $x_{rand}$, $\Delta$ x)
20     if ValidConnection($x_{nearest}$,$x_{new}$) then
21         G.add_node($x_{new}$)
22         $X_{near} \leftarrow$ FindNearNodes(G,$x_{new}$,$\gamma$)
23         $x_{min} \leftarrow x_{nearest}$
24         $cost_{min} \leftarrow$ Cost($x_{nearest}$)+Dist($x_{nearest}, x_{new}$)
25         UpdateCost($X_{near}$, $x_{new}$)
26         G.add_edge($x_{min}, x_{new}$) {Shortest path to $x_{new}$}
27         RewriteTree($X_{near}$, $x_{new}$)
28     if $x_{new} \in Goal$ then
29         $X_{dest} \leftarrow X_{dest} \cup x_{new}$
30 end
31 $x_{optimal}, path_{min} \leftarrow$ FindOptimalNode($G, \mathcal{X}_*$)
32 return $G$, $x_{optimal}$, $path_{min}$
```

randomly sampled vertex is included in the tree only if its connection to the nearest node meets the safety requirements specified in Eqs. 3.9-3.12. Additionally, edges between the new node and neighboring nodes within a radius γ are updated if more efficient connections are identified (lines 22-27). The algorithm identifies a mitigation path if the destination lies within the target region $\mathcal{X}_*$ (lines 28-29). Finally, it selects an optimal destination vertex and mitigation path (line 31) based on criteria such as minimizing distance to $\mathcal{X}_h$ or maximizing smoothness.

3.3.4 Response Control Action Generation

Once the target state trajectory is generated, it can be provided as input to a control algorithm, which can be either ML-based or model-based. This control algorithm then generates the final sequence of corrective or recovery control actions required to achieve the desired state trajectory.

Problem Statement: We model the task of generating a sequence of control actions as a context-specific multivariate sequence-to-sequence regression problem, as shown below:

$$\begin{aligned} & y_t = f(\bar{X}_{t+n}) = \hat{U}_{t+n-1}, \text{s.t.} \\ & \forall t' \in [t, t+n-1] : x_{t'} \overset{u_{t'}}{\rightarrow} x_{t'+1} \end{aligned} \tag{3.13}$$

Given the expected system state sequence $\bar{X}_{t+n} = \{\bar{x}_{t+1}, ..., \bar{x}_{t+n}\}$, the controller outputs a control action sequence $\hat{U}_{t+n-1} = \{\hat{u}_t, \ldots, \hat{u}_{t+n-1}\}$ that, if executed sequentially by the actuators, should transition the system to the expected states.

Domain Knowledge Specification: The objective of mitigation is to utilize the current system context and prediction of a potential hazard to determine a new corrective control action u_t^c from a set of control actions $\boldsymbol{u}^\rho$ to be sent to the actuators, ensuring the system transitions into the target or safe region. These context-specific corrective control actions can be specified based on domain knowledge. In this study, we construct the Hazard Mitigation Specification (HMS) by adhering to the formal framework presented in Section 2.3, utilizing a control-theoretic hazard analysis method known as STPA [14]. This procedure encompasses the following steps:

1. Define the hazards and adverse events of interest.
2. Identify the observable set of variables x_t of interest related to the hazards and decide on the possible transformations $\mu(x_t)$ and the sets $\rho(\mu(x_t))$ as completely as

Table 3.1: STL hazard mitigation specifications for APS.

Rule	Mitigation Action	STL Description of Safety Context
1	$G_{[t_0,t_e]}((F_{[0,t_s]}(u_2))$	$\mathcal{S}((BG > BGT \wedge BG' > 0) \wedge (IOB' < 0 \wedge IOB < \beta_1)))$
2	$G_{[t_0,t_e]}((F_{[0,t_s]}(u_2 \| u_4))$	$\mathcal{S}((BG > BGT \wedge BG' > 0) \wedge (IOB' = 0 \wedge IOB < \beta_2)))$
3	$G_{[t_0,t_e]}((F_{[0,t_s]}(u_2 \| u_4))$	$\mathcal{S}((BG > BGT \wedge BG' < 0) \wedge (IOB' > 0 \wedge IOB < \beta_3)))$
4	$G_{[t_0,t_e]}((F_{[0,t_s]}(u_2 \| u_4))$	$\mathcal{S}((BG > BGT \wedge BG' < 0) \wedge (IOB' < 0 \wedge IOB < \beta_4)))$
5	$G_{[t_0,t_e]}((F_{[0,t_s]}(u_2 \| u_4))$	$\mathcal{S}((BG > BGT \wedge BG' < 0) \wedge (IOB' = 0 \wedge IOB < \beta_5)))$
6	$G_{[t_0,t_e]}((F_{[0,t_s]}(u_3))$	$\mathcal{S}((BG < BGT \wedge BG' < 0) \wedge (IOB' > 0 \wedge IOB > \beta_6)))$
7	$G_{[t_0,t_e]}((F_{[0,t_s]}(u_1 \| u_3))$	$\mathcal{S}((BG < BGT \wedge BG' < 0) \wedge (IOB' < 0 \wedge IOB > \beta_7)))$
8	$G_{[t_0,t_e]}((F_{[0,t_s]}(u_1 \| u_3))$	$\mathcal{S}((BG < BGT \wedge BG' < 0) \wedge (IOB' = 0 \wedge IOB > \beta_8)))$
9	$G_{[t_0,t_e]}((F_{[0,t_s]}(\neg u_3))$	$\mathcal{S}((BG > BGT \wedge IOB < \beta_9)))$
10	$G_{[t_0,t_e]}((F_{[0,t_s]}(u_3))$	$\mathcal{S}((BG < \beta_{10})))$

* BGT: BG target value, IOB: Insulin on board; $BG' = dBG/dt$, $IOB' = dIOB/dt$;
* $u_{1,2,3,4}$:decrease_insulin, increase_insulin, stop_insulin, keep_insulin;

possible.

3. List all the combinations of $\rho(\mu(x_t))$ and control action u_t and identify the combinations that might result in transitions to a hazardous region $\mathcal{X}_h$. The unknown boundaries β_i can be learned from a population dataset using off-the-shelf optimizers (e.g., Newton's method) [47,53].
4. For each unsafe context in step 3, find all control actions $u_t^c \in U$ such that $(\rho(\mu(x_t)), u_t^c) \mapsto \mathcal{X}_*$ and add these tuples $(\rho(\mu(x_t)), u_t^c)$ to HMS for that context.
5. Formalize the generated HMS into STL format.

An example of a generated STL hazard mitigation specification for APS is presented in Table 3.1. This table specifies a set of STL rules designed to mitigate potential hazards that could lead to hyperglycemia or hypoglycemia events. For instance, the first rule indicates that if the system context shows that blood glucose (BG) is higher than the target and continuously rising, while insulin on board (IOB) is below a threshold β_1 and continuously decreasing, then an *increase_insulin* action u_2 should be initiated by the controller within a time frame t_s to prevent a hyperglycemia event. It's crucial for this safety property to be maintained throughout the entire operational period.

It's important to acknowledge the semantic gap between the high-level, human interpretable state variables $\mu(x)$ in the Hazard HMS, such as the rate of blood glucose

(BG) change, and the sensor measurements (e.g., BG values from Continuous Glucose Monitoring or CGM), as well as between the high-level control actions u_t^c (e.g., *increase insulin*) and the low-level control commands sent to the actuator (e.g., the amount of insulin dose). To bridge this gap during actual implementation, an additional step is required to infer the values of state variables in the transformed state space based on sensor measurements (e.g., by calculating the derivative of BG values to determine the rate of BG change) and to derive the high-level control commands based on Machine Learning (ML) predictions, utilizing the transformation function $h(\cdot)$ (refer to Eq. 3.14). These transformations facilitate the alignment of the estimated states and control actions with the logic formulas specified in the HMS, enabling the verification of the satisfaction of any safety properties (e.g., Rules 1-5 in Table 3.1 will be assessed for control action u_2).

Data and Knowledge Integration: In this work, we integrate the domain-specific knowledge on context-dependent mitigation actions with state variable and control output data traces collected from closed-loop CPS simulation or real operation to train an RNN for control action generation (referred to as *ActNet*). We encode the logic formulas generated for HMS as a custom loss function that penalizes the RNN model during the training process if the ML prediction fails to align with the specified mitigation action properties:

$$\mathcal{L} = \eta\mathcal{L}(U,\hat{U}) + (1-\eta)k(2\sigma(\sum_{h(y_t)\notin u_t^c} \omega_i) - 1) \tag{3.14}$$

In the provided equation, $h(\cdot)$ represents a transformation function (e.g., derivation) that converts the predicted control command y_t into a discrete high-level control action, as discussed earlier. The parameter w serves as a weight parameter, determining the influence of each logic rule on the training process. Furthermore, $\sigma(\cdot)$ denotes a sigmoid function that maps the degree of satisfaction of the ML outputs with the STL formulas to the range [0,1]. Lastly, k serves as a scaling factor, which adjusts the sigmoid function to a similar range as the original loss $\mathcal{L}(U,\hat{U})$ (e.g., mean squared error).

3.3.5 Safety Engine Implementation

The complete hazard prediction and mitigation procedure is outlined in Algorithm 3. Our two-level hazard prediction framework comprises (i) a long-term prediction model (*PredNet-l*) capable of forecasting a distant future state sequence, ensuring adequate time to anticipate and mitigate potential hazards (line 36), and (ii) a short-term prediction model (*PredNet-s*) with higher accuracy than the long-term model, guaranteeing the detection of any hazards (line 30). The control system will trigger an alert and transition to the *Mitigation* mode whenever any predicted system state sequence intersects the unsafe region $\mathcal{X}_h$ (lines 31-32 and lines 37-38). To optimize computation time (refer to Section 3.4.5.8), *PredNet-l* is activated only when *PredNet-s* does not anticipate any hazards in the near future.

An optimal mitigation path is determined based on the available mitigation budget (line 43). If potential hazards are identified by the long-term hazard prediction model (which has a larger mitigation budget), strict constraints are employed to formulate the mitigation path. These constraints ensure adherence to smoothness and safety criteria throughout the mitigation process (lines 36-40). Conversely, if a hazard is not predicted by the long-term model but is identified by the short-term model, relaxed constraints (e.g., wider ranges of α_i^j in Eq. 3.10) are applied to facilitate rapid mitigation within the short mitigation deadline (lines 31-34). These prediction models and mitigation path generation strategies complement each other to enhance mitigation performance. Subsequently, a sequence of corrective control actions U^c is derived by inputting the mitigation path into the mitigation action generation model (*ActNet*) (line 47).

When operating in the *Mitigation* mode (line 2), the *GetAction* function sequentially selects a mitigation action u^c_{kMit} from the set U^c (line 14), which is then executed by the actuators (line 20). To balance accuracy with computational overhead, the mitigation path and control action sequences are updated only when the error between the actual

Algorithm 3: Hazard Prediction and Mitigation

```
Function Mitigation():
    if $MitEnable$ then
        if $x_t \in \mathcal{X}_*$ or $kMit \geq \mathcal{D}$ then
            $MitEnable \leftarrow False$ {Stop }
            $kMit \leftarrow 0$
            if $x_t \notin \mathcal{X}_*$ then
                RaiseAlert("Mit. Failure!")
        else
            if $kMit > 0$ and $err(\bar{X}_{[kMit-1]}, x_t) > \theta$ then
                $\bar{X} \leftarrow SC\_RRT^*(x_t, \mathcal{D} - kMit, QuickMit)$
                $U^c \leftarrow ActNet(\bar{X})$ {Update}
                $kMit \leftarrow 0$
                $\mathcal{D} \leftarrow \mathcal{D} - kMit$
            $u \leftarrow GetAction(U^c, kMit)$ {Replace}
            $kMit \leftarrow kMit + 1$
        end
    else
        $u \leftarrow u_t$
    end
    Actuate($u$)
return
$x \leftarrow$ System State
$u \leftarrow$ Control Action to the Actuator
$Hazard \leftarrow False$ {Hazard Prediction Flag}
for each control cycle $t$ do
    $X_t \leftarrow UpdateState(x_t)$
    $U_t \leftarrow UpdateAction(u_t)$ {Original Control Action}
    if $MitEnable{=}{=}False$ then
        $kMit \leftarrow 0$
        $\hat{x}_{[t+1:s]} \leftarrow PredNet\text{-}s(X_t, U_t, s)$ {Short-term}
        if $\{\hat{x}_{[t+1:s]}\} \cap \mathcal{X}_h$ then
            $MitEnable \leftarrow True$
            $\mathcal{D} \leftarrow Deadline(\hat{x}_{[t+1:s]})$
            $QuickMit \leftarrow True$
        else
            $\hat{x}_{[t+1:l]} \leftarrow PredNet\text{-}l(X_t, U_t, l)$ {Long-term}
            if $\{(\hat{x}_{[t+1:l]}\} \cap \mathcal{X}_h$ then
                $MitEnable \leftarrow True$
                $\mathcal{D} \leftarrow Deadline(\hat{x}_{[t+1:l]})$
                $QuickMit \leftarrow False$
        end
        if $MitEnable$ then
            $\bar{X} \leftarrow SC\_RRT^*(x_t, \mathcal{D}, QuickMit)$
            if $\bar{X} == \emptyset$ then
                RaiseAlert("Mit. Path Not Found!")
                break
            $U^c \leftarrow ActNet(\bar{X})$ {Corrective Action Seq.}
    end
    Mitigation()
end
```

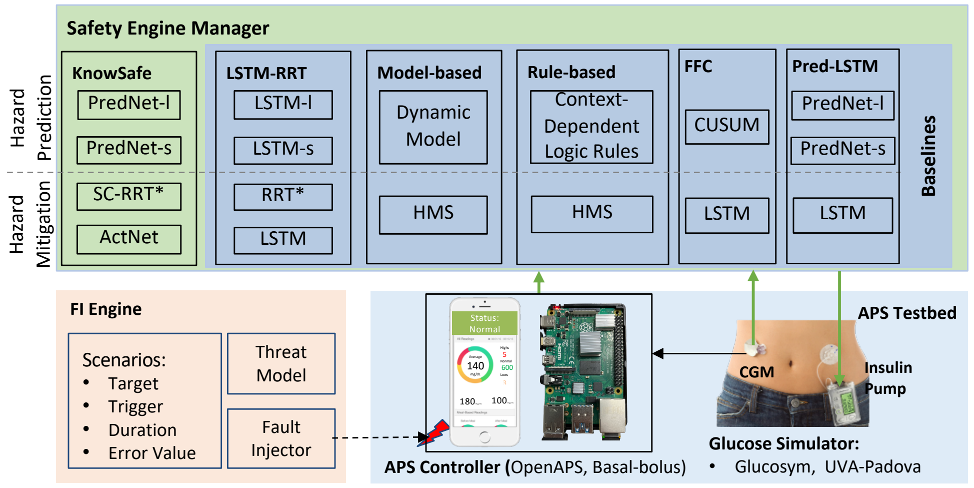


Figure 3.4: Experimental platform with different safety engines, integrated with an APS controller running on a PC/cellphone or an embedded device (e.g., Raspberry Pi 4), a glucose simulator, and a software fault injection engine.

state x_t and the expected state in the mitigation path $\bar{X}$ exceeds a predefined threshold θ (line 9). The *Mitigation* mode remains active until the system returns to the target region $\mathcal{X}_*$ or the mitigation budget $\mathcal{D}$ has been depleted (lines 3-5). Users (patients) or physicians will be alerted to any mitigation failures, such as when no safe path is found (line 45) or when the budget is exhausted without reaching a safe state (line 7).

3.4 Experimental Evaluation

We develop an open-source simulation environment (see Fig. 3.4) that integrates the closed-loop simulation of two example APS controllers with an adverse event simulator to evaluate different safety engines. We run the experiments with both APS controllers and simulators on an x86_64 PC with an Intel Core i9 CPU @ 3.50GHz and 32GB RAM running Linux Ubuntu 20.04 LTS. We use TensorFlow v.2.5.0 to train our ML models and Scikit-learn v.1.1.1 for data pre-processing and experimental evaluation.

3.4.1 Experimental Platform

As illustrated in Fig. 3.4, the APS testbed incorporates two widely-used APS controllers (OpenAPS [77] and Basal-Bolus [78]) along with two distinct patient glucose simulators: Glucosym [79] and UVA-Padova Type 1 Diabetes Simulator [26] (introduced in Section 2.8). More details about the APS testbed design are provided in Appendix A.1.

Each experiment involves the patient simulator interacting with the APS controller for 150 iterations, equivalent to about 12.5 actual hours. Each iteration in the simulation represents 5 minutes in the real APS control system. Simulations commence with the patient having varying initial glucose values within the normal range of 70 to 180 mg/dL, with no additional meals or exercise during the simulation period, emulating a scenario of the patient at nighttime after consuming dinner. To accommodate for inter-patient variability, each system version (without a safety engine and with each different safety engine) undergoes evaluation using ten patient profiles in each glucose simulator.

3.4.2 Adverse Event Simulation

We conduct closed-loop simulations of a glucose simulator and an APS controller using a source-level fault injection (FI) engine. This engine directly perturbs the values of the controller's state variables within their acceptable ranges over a random period. This simulation is aimed at simulating the effect of accidental faults or attack scenarios introduced in Section 2.8.3 (refer to Table 2.2). We simulate two categories of attacks: (1) **Availability attacks** involve setting the control output to zero (*Min* scenario for the *Rate* variable in Table 3.2) or maintaining the same sensor measurements after the k_{th} control cycle for m steps ($x_{k:k+m} = x_k$, referred to as the *Hold* scenario in Table 3.2); and (2) **Integrity attacks** entail adding a bias b to the sensor readings ($x_k = x_k + b$). Various scenarios are considered, including *Add*, *Subtract*, *Max*, and *Min* scenarios (as outlined in Table 3.2).

Table 3.2: Adverse event simulation experiments per patient.

Target State Variables	**Scenario**	**Attack Value**		**No. Sim.**
		BG	**Rate**	
Blood Glucose (BG)/ Insulin Output (Rate)	Hold	Repeat	Repeat	63
	Add	[32,64]	[0.5,1]	126
	Subtract	[32,64]	[0.5,1]	126
	Max	175	2	63
	Min	80	0	63

For each FI scenario, the FI engine determines (i) the target state variable, (ii) the trigger condition for activating the fault, (iii) the duration of the fault, and (iv) the error values to be injected. In this work, we employ a random strategy that selects from several different start times and durations, uniformly distributed within the entire simulation period, to inject the faults. This results in 882 FI simulations for each patient (3 start times $\times$ 3 durations $\times$ 7 initial BG values $\times$ 14 attack values, see the last column of Table 3.2), and a total of 17,640 simulations for all 20 patients across both simulators. This equates to 25.52 years of simulation data (17,640 $\times$ 12.5 hours) used for training and testing various safety engines. The details of each FI scenario are provided in Table 3.2.

3.4.3 Adversarial Training

We utilize both normal and hazardous data to train hazard prediction models, aiming to enhance robustness. For evaluating the trained state and hazard prediction models, we treat the entire simulation trace as a sample. It is labeled as hazardous if any state sequence in that trace overlaps the unsafe region (e.g., BG values higher than 180 mg/dL or less than 70 mg/dL in APS).

For training and testing response control action generation models, we utilize a sequence of collected control actions $u_{t:t+n}$ as the ground truth values. The input to the models consists of the expected or target system states $x_{t+1:t+n}$ upon sequential execution of the control actions.

Our FI engine results in a total of 3,230 (38.3%) simulations with hazards in the Glucosym simulator and 3,315 (37.6%) simulations with hazards in the UVA-Padova simulator. These are employed for training and testing each ML model for hazard prediction and mitigation.

We employ a 4-fold cross-validation setup for training and testing each patient-specific ML model. Specifically, for each patient, we train separate ML models for state prediction and mitigation action generation. This is done using data from 75% of the simulation traces, with the models tested on the remaining 25% of simulation traces.

To streamline model complexity and conserve computational resources, we minimize the number of layers in the neural network. After exploring various model architectures, the most effective model we obtained was a two-layer (128-64 units) stacked LSTM.

3.4.4 Metrics

We introduce the following metrics to evaluate the performance of the proposed methodology.

- ***Prediction Accuracy*** is assessed using the root mean squared error (RMSE) between the predicted or generated trajectories and the ground truth trajectories. Additionally, we employ standard binary metrics including false positive rate (FPR), false negative rate (FNR), and F1 score to evaluate the accuracy of hazard prediction and the activation of mitigation actions. Performance evaluation is conducted by considering the entire trajectory of each simulation as a sample.

- ***Reaction Time*** measures the timeliness of hazard prediction and indicates the maximum time budget to mitigate and prevent potential hazards. It is the time difference between when the prediction network detects the hazard and when the system enters a hazardous state.

- ***Mitigation Path Planning Efficiency*** is measured by evaluating the performance of each algorithm in finding an optimal mitigation path using the following two metrics:

  - ***Convergence Rate*** is the percentage of simulations in which a mitigation path ends up in a final state in the safe region.
  - ***Satisfaction Rate*** is the percentage of simulations with a valid mitigation path that satisfies safety constraints.

- ***Mitigation Outcome*** metrics evaluate the overall performance of the safety engine (the hazard prediction and mitigation pipeline):

  - ***Mitigation Success Rate (MSR)*** is calculated as the percentage of hazardous simulations in which the system transitions into the unsafe region without mitigation but after implementing the mitigation actions returns to the target region.
  - ***Out-of-Safe-Range Rate*** is the percentage of simulations with state variable values falling outside of the safe range (e.g., higher than 180 mg/dL or less than 70 mg/dL in APS). This metric measures the remaining hazard rate even with the mitigation algorithm, the summation of which is the complementary set of the *Mitigation Success Rate*.
  - ***Max Deviation*** measures the maximum distance from the safe region boundaries. This metric evaluates the risk of transitioning to an unsafe state with or without safety engine.
  - ***New Hazard Rate*** represents the percentage of non-hazardous simulations in which new hazards are introduced due to false alarms and performing unnecessary mitigation.

The final results are reported by calculating the average value of each metric over all cross-validation folds across all patients.

Table 3.3: Performance of customized classification models.

Simulator	Model	FPR	FNR	ACC	F1 Score
Glucosym	MLP	0.02	0.07	0.97	0.89
	LSTM	0.04	0.06	0.96	0.81
	MLP-Custom	0.02	0.05	0.98	0.91
	LSTM-Custom	**0.00**	0.23	0.97	0.86
T1DS 2013	MLP	<0.01	0.56	0.94	0.71
	LSTM	<0.01	0.06	0.99	0.95
	MLP-Custom	0.01	0.27	0.96	0.82
	LSTM-Custom	**0.00**	0.17	0.98	0.90

3.4.5 Results

3.4.5.1 Hazard Prediction Accuracy

(1) *Assessment of Customized Classification Models:* As shown in Table 3.3, the MLP-Custom model achieved a 1.9% and 16.2% improvement in F1 score and 28.6% and 38.6% reduction in FNR while keeping FPR low for the Glucosym and T1DS2013 simulators, respectively. We observed a similar improvement in the overall performance of the LSTM-Custom model for both the Glucosym and T1DS2013 simulators using at least one prediction accuracy metric.

(2) *Assessment of Customized Regression Models:* Table 3.4 illustrates the performance of the proposed prediction models (*PredNet*) for system state and hazard prediction, as well as mitigation deadline estimation in both glucose simulators, in comparison to the baseline long short-term memory (LSTM) models with identical architecture to *PredNet*. We use "l" and "s" to distinguish between ML models with long-term and short-term prediction windows (e.g., 24 and 6 control cycles in this study), respectively. The final results are aggregated by computing the average value of each metric across all cross-validation folds and all patients.

Using simulation data from Glucosym, we observe that ML models trained with domain knowledge guidance achieve superior prediction accuracy, with up to 37.7% RMSE

Table 3.4: Performance of each ML model in predicting system states and estimating deadlines of mitigation averaged over 8,820 simulations in Glucosym and UVA-Padova simulators, respectively. RMSE is measured in mg/dL for state estimation and *Iteration* (representing 5 minutes in actual APS) for deadline estimation.

Simulator	**Model**	**State**	**Hazard Prediction**			**Deadline**
		RMSE	**FNR**	**FPR**	**F1**	**RMSE**
Glucosym	LSTM-l	2.34	0.037	0.091	0.909	0.89
	LSTM-s	0.61	0.008	0.017	0.982	0.17
	PredNet-l	**1.46**	**0.014**	**0.061**	**0.943**	**0.64**
	PredNet-s	**0.38**	**0**	**0.003**	**0.997**	**0.13**
UVA-Padova	LSTM-l	7.66	0.213	0.091	0.899	2.24
	LSTM-s	3.14	0.045	0.082	0.951	0.21
	PredNet-l	**7.20**	**0.015**	**0.039**	**0.961**	**2.07**
	PredNet-s	**2.85**	**0**	**0.012**	**0.993**	**0.20**

(0.38 vs. 0.61) in state prediction and 28.1% lower RMSE (0.64 vs. 0.89) in deadline estimation.

Although the *PredNet*s exhibit only marginally better F1 scores in hazard prediction (1.5%-3.8% improvement), they achieve a zero FNR, which is crucial for hazard prediction and mitigation, while maintaining an FPR that is up to 82.3% lower (0.003 vs. 0.017).

In tests conducted on the UVA-Padova simulator, we observe similar performance enhancements of combined knowledge and data-driven ML models in hazard prediction (4.4%-6.9%) and system state prediction and deadline estimation. This suggests the advantage of integrating knowledge with data-driven approaches in predicting hazards and achieving consistent performance across different simulators and patients. However, the overall prediction error is higher in the UVA-Padova simulator compared to the Glucosym simulator for all models. This discrepancy may be attributed to the inclusion of a noise model for sensor measurements in the UVA-Padova simulator [133].

In Table 3.4, we observe that PredNet-s achieves a notable reduction of up to 74.0% (0.38 vs. 1.46) and 90.3% (0.20 vs. 2.07) in RMSE for state and deadline prediction, respectively, compared to PredNet-l. Additionally, PredNet-s exhibits at least 69.2% lower FPR (0.012 vs. 0.039) in predicting the occurrence of hazards while maintaining

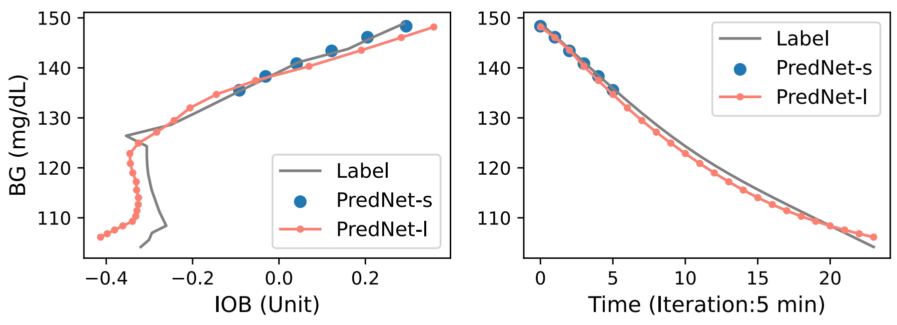


Figure 3.5: Example state trajectories predicted by both long-term and short-term hazard prediction models.

a zero false negative rate. This highlights the advantage of employing two-level hazard prediction models, leveraging the high accuracy of PredNet-s models to avoid missing any positive cases and the extended prediction window of PredNet-l models to proactively mitigate potential hazards effectively.

An example of the state trajectory and the predictions of PredNets is illustrated in Fig. 3.5. PredNet-s predictions closely overlap with the ground-truth state trajectory, while the long-term predictions by PredNet-l also approximate the ground truth well.

To further assess the performance of PredNet in real-world APS and mitigate the simulation-to-real gap, we train and evaluate the combined knowledge and data-driven ML models for system state sequence prediction using a publicly available clinical trial dataset, PSO3 [134]. Experimental findings indicate that our PredNet achieves a slightly smaller RMSE compared to ML models developed in prior research (6.349 vs. 7.187 mg/dL on average across the same patient data) [135]. It's noteworthy that we do not evaluate the hazard mitigation approach on actual APS as it necessitates runtime execution in a closed-loop system, which would pose safety concerns when implemented with actual patients.

3.4.5.2 Evaluation of Mitigation Path Generation

Table 3.5 displays the outcomes of the mitigation path generation algorithm, SC-RRT*, juxtaposed with a baseline RRT* algorithm that overlooks application-based constraints

Table 3.5: Performance of each mitigation path generation algorithm (averaged over all the simulations). A unit of prediction length or path length represents 5 minutes in actual APS. *Conv. Trials* represents the number of trials each RRT algorithm takes to converge to a valid path.

Simulator	**Prediction Model**	**Algorithm**	**Conv. Trials**	**Path Len**	**Satisfaction Rate**
Glucosym	PredNet-s	SC-RRT*	174	15	89.4%
		RRT*	32	10	6.4%
	PredNet-l	SC-RRT*	194	25	85.8%
		RRT*	41	14	3.4%
UVA-Padova	PredNet-s	SC-RRT*	245	10	93.5%
		RRT*	78	8	31.9%
	PredNet-l	SC-RRT*	167	22	94.7%
		RRT*	50	17	32.0%

Table 3.6: Domain-specific constraints for APS. (BG is in mg/dL and IOB is in $unit$).

No.	**Description**	**Constraint**	**No.**	**Description**	**Constraint**
1	dBG/dt	[-5, 3]	3	$dIOB/dt$	[-0.1, 0.1]
2	d^2BG/dt^2	[-2.5, 2.5]	4	d^2IOB/dt^2	[-0.05, 0.05]

(as delineated in Eq. 3.9-3.10 and exemplified in Table 3.6).

Both algorithms successfully converge to a solution across all experiments. However, the baseline algorithm exhibits a notably lower satisfaction rate (the last column of Table 3.5). This discrepancy arises because the baseline algorithm fails to account for physical constraints or safety requirements during path generation.

An illustrative example of mitigation paths generated by both algorithms is presented in Fig. 3.6. It is evident that the baseline RRT* (depicted by orange dots) produces sharp turns early in the mitigation process, which could be unattainable due to significant lag in the biological blood glucose (BG) process or could cause patient discomfort owing to abrupt BG level changes. In contrast, the proposed SC-RRT* algorithm (marked by blue dots) generates smoother trajectories for hazard mitigation by considering rate and trend of changes along with physiological constraints.

Although the SC-RRT* algorithm necessitates more trials to converge to an optimal

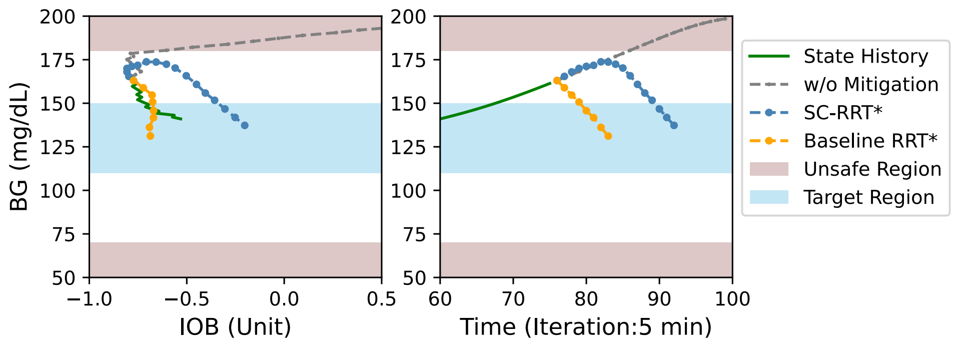


Figure 3.6: Example mitigation paths by SC-RRT* and baseline RRT* algorithms in the state space (Left) and time space (Right).

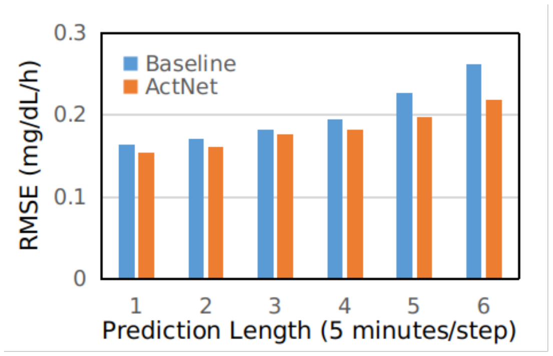


Figure 3.7: Performance of control action generation (ActNet).

path, the average time required is much smaller than the length of a control cycle, thereby having minimal impact on system operation. Further analysis of time overhead for each model is discussed in Section 3.4.5.8.

Additionally, Table 3.5 reveals that mitigation paths generated based on hazards inferred by the PredNet-s model exhibit shorter lengths (averaged over all patients in each simulator) compared to those generated based on PredNet-l predictions. This highlights the efficacy of PredNet-s in ensuring valid mitigation paths within short mitigation deadlines, complementing PredNet-l's role in ensuring smooth state transitions and enhancing user experience.

3.4.5.3 Evaluation of Response Action Generation

Fig. 3.7 illustrates the performance of the response action generation models, with and without the integration of Hazard Mitigation Specification (HMS) knowledge, in accurately reconstructing the control action sequence and replicating the controller dynamics.

It's evident that the ML model trained using the proposed custom loss function (Eq. 3.14), referred to as ActNet, consistently maintains a lower RMSE compared to the baseline LSTM model across all prediction lengths. This outcome underscores the advantage of combining knowledge and data in accurately generating mitigation actions.

We also notice that the RMSE of ActNet is slightly lower than that of the baseline for one-step prediction. However, as the prediction length increases, the disparity between ActNet and the baseline widens. This is attributed to the fact that integrated domain knowledge is unaffected by data and prediction length.

This observation underscores the more apparent advantage of integrating domain knowledge on context-specific mitigation actions with ML models for enhancing sequential prediction accuracy compared to single value regression. It ensures the maintenance of a stable and low RMSE while reducing performance degradation.

Furthermore, the integration of domain knowledge (Hazard Mitigation Specification, HMS) contributes to enhancing the explainability of black-box ML models in safety-critical CPS. It provides a rationale for selecting a corrective control action under a given context and can be utilized for runtime verification of ML model outputs.

3.4.5.4 End-to-End Safety Engine Evaluation

We integrate the proposed safety engine (the pipeline of PredNet, SC-RRT*, ActNet models) as well as several state-of-the-art baselines with the controllers in our closed-loop testbed and compare their performances.

(1) Baselines: We contrast our integrated knowledge and data-driven strategy, *KnowSafe*, with approaches solely driven by knowledge or data. This comparative analysis involves

devising various baseline methods, such as a rule-based approach that employs identical domain knowledge concerning constraints and mitigation actions as *KnowSafe*, a model-based approach, and ML-based methods with similar architectures.

The **rule-based** baseline comprises two components: (i) An anomaly detection algorithm crafted from context-dependent specifications of unsafe control actions, and (ii) A mitigation algorithm formulated based on the generated HMS in Table 3.1. The detection rules determine whether a control action issued by the controller in a given context might lead to a transition into the unsafe region. The mitigation rules outline corrective actions to be implemented in each context to maintain safety within the safe region.

We implement existing state-of-the-art defense solutions (e.g., CI [20] or SAVIOR [46]) as a **model-based** baseline, employing a dynamic model of the physical system (patient physiology), known as the Bergman & Sherwin model [102]. This model estimates the potential BG value (BG_{t+1}) after executing the pump's command (u_t) on the patient's current state (BG_t). If the predicted BG value exceeds the patient's normal range ([70,180] mg/dL, as defined by medical guidelines), a mitigation algorithm, similar to the rule-based mitigation, issues corrective actions, replacing the control action by the controller until the system state returns to the target range.

Additionally, we develop two solely data-driven baseline models. The first integrates LSTM models and RRT* with the same architecture as *KnowSafe* but without the integration of domain-specific knowledge (referred to as **LSTM-RRT**). For the LSTM models, this is equivalent to setting η to 1 in Eq. 3.6 and Eq. 3.14. The second data-driven model directly maps sensor measurements to the control output y_{ML} using a single regression model, based on a feed-forward control (**FFC**) method proposed by [33] for unmanned vehicles. In this approach, when the accumulated error between the ML output and the controller output exceeds a preset threshold, the mitigation mode is activated, and the controller output is replaced by the ML output to be delivered to the actuator. (It's noted that since these previous works are not directly applicable to APS, we re-implement their

Table 3.7: Mitigation performance and time overhead of KnowSafe vs. baselines. The severity of hypoglycemia (BG<70 mg/dL) increases with decreasing BG value, including mild (54 mg/dL<BG<70 mg/dL) and more severe (BG<54 mg/dL) categories.

Method	**No. Sim.**	**Out-of-Safe-Range Rate** (mg/dL)			**MSR**	**Time Overhead (ms)**	
		BG>180	**BG<70**	**BG<54**		Normal Mode	Mit. Mode
KnowSafe	6,545	**428 (6.5%)**	**44 (0.7%)**	**0**	**92.8%**	46.5	54 or 0
LSTM-RRT	6,545	2492 (38.1%)	603 (9.2%)	66 (1%)	52.7%	46.2	81.5 or 0
Rule-based	6,545	1421 (21.7%)	1792 (27.4%)	370 (5.7%)	50.9%	0.1 (106.5 μs)	0.1 (121.5 μs)
Model-based	6,545	1481 (22.6%)	3685 (56.3%)	716 (10.9%)	21.1%	87.9	87.9
FFC	6,545	499 (7.6%)	459 (7.0%)	39 (0.6%)	85.4%	28.8	28.8
Pred-LSTM	6,545	467 (7.1%)	285 (4.4%)	**0**	88.5%	46.5	28.7

high-level pipeline while customizing their models and parameters.)

To assess the effectiveness of the proposed hazard mitigation module, we design another hybrid baseline, **Pred-LSTM**, which employs the same prediction module (Pred-Net) as *KnowSafe* but utilizes a single LSTM model (with HMS integration) to generate corrective control actions (without the path planning part). Further details of each baseline are illustrated in Fig. 3.4.

(2) Mitigation Outcome: We conduct a re-run of all hazardous simulations (6,545 in both simulators) with different safety engines (*KnowSafe* vs. baselines) and provide their performance in mitigating hazards and maintaining the system inside the safe region in Table 3.7.

The rule-based baseline successfully mitigates 50.9% of hazards, as indicated by the mitigation success rate (MSR), but it fails to maintain the remaining half of the simulations within the safe region. Similarly, the LSTM-RRT baseline effectively prevents 52.7% of hazards, but 3,095 simulations fall outside the safe region. This discrepancy arises because, while the rule-based strategy possesses contextual knowledge, the generated high-level mitigation action fails to infer specific quantitative values for mitigating potential hazards. Conversely, the ML baseline offers more precise mitigation actions but relies on a black-box data-driven model that may violate HMS rules under certain contexts. Therefore, by integrating knowledge with data, KnowSafe overcomes the shortcomings of either approach and leverages the strengths of both methods. It achieves a mitigation success rate at least 41.9% higher than solely rule-based and ML-based (LSTM-RRT) baselines, while also minimizing the number of simulations outside the safe range.

Additionally, we observe that *KnowSafe* outperforms the model-based and FFC baselines by achieving the highest MSR and the fewest simulations with BG outside the target range. Notably, BG falls below 54 mg/dL in 39 and 716 simulations with FFC and model-based mitigation, respectively, which could lead to severe hypoglycemia and serious complications (e.g., seizure, coma, or even death). In contrast, *KnowSafe* maintains BG above 54 mg/dL for all simulations. Although BG drops below 70 mg/dL in 44 simulations with *KnowSafe*, the percentage is significantly lower than the baselines and poses fewer concerns compared to cases with BG below 54 mg/dL.

The superior performance of our proposed mitigation approach over these baselines may be attributed to our approach's emphasis on early hazard prediction and mitigation by estimating the possibility of the system state entering the unsafe region. In contrast, approaches like the FFC and model-based baselines detect hazards after they occur or wait until the error between the ML predictions and the actual system states or outputs exceeds a noticeable threshold, thus wasting the limited mitigation time budget on hazard detection and potentially leading to mitigation/recovery failure.

While Pred-LSTM shares the same prediction module as *KnowSafe* and benefits from early hazard prediction, its overall MSR is 4.3% lower. This outcome underscores the advantage of our hazard mitigation pipeline (SC-RRT* and ActNet) over a single ML model that directly outputs a control action based on current system states. SC-RRT* can find an optimal mitigation path, while Pred-LSTM can only generate a single-step mitigation action at each control cycle. Since Pred-LSTM shares the same prediction module as *KnowSafe*, we do not compare its performance in the following sections.

Table 3.8 further illustrates the maximum deviations of BG from the safe region boundaries in each simulation, with and without the proposed safety engine. The maximum deviation from the lower bound of the safe region decreases from 49.6 mg/dL to 4.3

Table 3.8: Average BG value and max deviations in the adverse event simulations with no mitigation or with KnowSafe.

Metric	**Max Deviation** (mg/dL)		**Average BG** (mg/dL)
	Above 180	**Below 70**	
Simulations w/o Mitigation	175.3	49.6	97.3
Simulations with KnowSafe	21.6	4.3	128.9

mg/dL (91.3% reduction). Similarly, the maximum deviation from the upper bound of the safe region decreases from 175.3 mg/dL to 21.6 mg/dL (87.6% reduction). Consequently, *KnowSafe* maintains a BG level within the range of [65.7, 201.6] with an average value of 128.9 mg/dL inside the target range, effectively averting all hypoglycemia and hyperglycemia hazards.

3.4.5.5 Timeliness

In Fig. 3.8, we present the timeliness of each mitigation strategy in predicting potential hazards, measured by the average reaction time. Our observations are as follows:

- The model-based baseline exhibits the lowest reaction time and MSR (see Table 3.7), underscoring the importance of sufficient reaction time for effective mitigation/recovery.
- The LSTM-RRT baseline achieves a reaction time similar to *KnowSafe* and a 1.4 times higher MSR than the model-based approach, highlighting the benefits of early hazard prediction. However, the LSTM-RRT baseline still requires refinement to mitigate the remaining 49.1% hazards, indicating that early detection alone is insufficient to ensure mitigation success.
- The rule-based baseline demonstrates a smaller reaction time than the LSTM-RRT approach but achieves a similar MSR, suggesting that context-awareness enhances mitigation performance.
- The FFC baseline exhibits the longest reaction time by generating alerts throughout

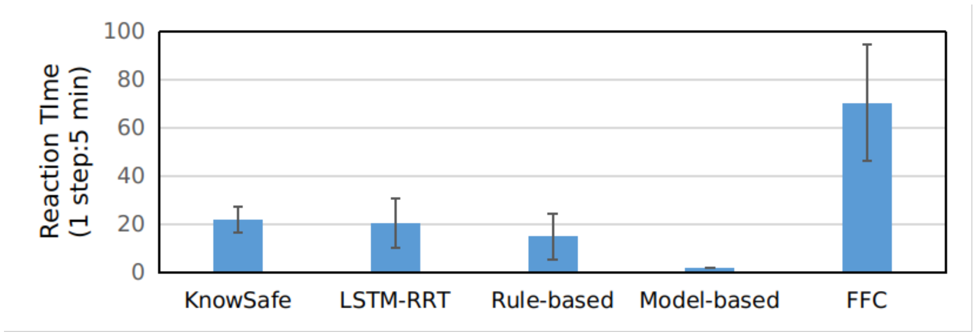


Figure 3.8: Reaction time of each mitigation strategy.

the entire simulation period, potentially causing user inconvenience and unnecessary mitigation efforts. We will further analyze false positive rates in the following subsection.

- *KnowSafe* maintains a stable and adequate reaction time while achieving the highest MSR, highlighting *the advantage of combining domain-specific knowledge with data to achieve both context-awareness and early hazard prediction for successful mitigation/recovery.*

3.4.5.6 False Positives

We also assess the repercussions of inadvertent mitigation actions triggered by false alarms of each mitigation strategy by testing them with all non-hazardous simulations (11,095 in both simulators).

In Fig. 3.9, we observe that the FFC baseline is falsely activated for 96.9% of simulations, resulting in 21.3% new hazards. This highlights the limitations of relying solely on a single ML model for accurately detecting and mitigating hazards. Comparatively, the LSTM-RRT baseline achieves a lower FPR and new hazard rate than the model-based and FFC baselines, indicating the advantages of the proposed hazard prediction and mitigation pipeline. While the rule-based safety engine exhibits twice the false positives compared to the LSTM-RRT approach, it maintains a much lower new hazard rate, demonstrating the minimal risk associated with the generated HMS even with false activation.

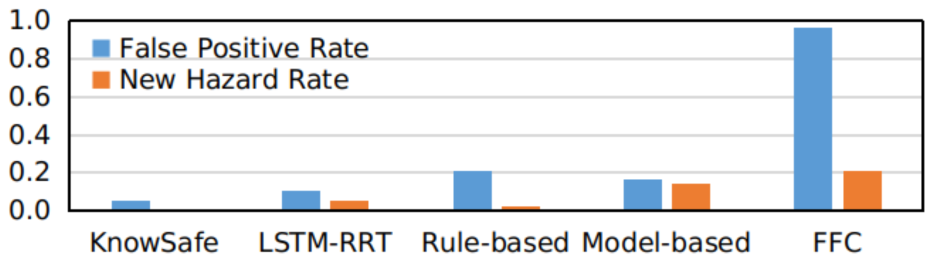


Figure 3.9: Performance of different safety engines in 11,095 non-hazardous simulations.

In contrast, *KnowSafe* maintains the lowest FPR (5.4%) without introducing any new hazards, emphasizing *the benefits of integrating domain knowledge with data-driven techniques to reduce the FPR and mitigate the risk of introducing new hazards in the hazard prediction and mitigation process.*

3.4.5.7 Mitigation of Stealthy Attacks

To assess the efficacy of the proposed mitigation strategy against stealthy attacks, we consider a sophisticated attacker who is familiar with the mitigation mechanism, including the ML architecture and parameters. We repeat the experiments outlined in Table 3.2, wherein the attacker refrains from executing the attack action (or injecting faults) to remain undetected, particularly when such action would trigger mitigation alarms if initiated. We compare the performance with a model-based baseline (similar to existing solutions CI and SAVIOR) and an ML-based baseline (e.g., LSTM-RRT introduced in Section 3.4.5.4). However, we exclude testing the FFC baseline due to its elevated FPRs.

In Table 3.9, it is evident that *KnowSafe* achieves a 100% success rate under stealthy attacks, maintaining BG values within the safe range of [70,180] mg/dL in all simulations conducted with both the Glucosym and UVA-Padova simulators.

In comparison, the model-based approach fails to mitigate the stealthy attack in 5,026 simulations (76.8% of 6,545 hazardous cases in Table 3.7), with a maximum deviation of 74.6 mg/dL above 180 mg/dL and 26.5 mg/dL below 70 mg/dL. This failure stems from

Table 3.9: Performance of each mitigation strategy against stealthy attacks.

Method	No. Simulation Out of Safe Range [70,180] mg/dL	Max Deviation (mg/dL)	
		Above 180	Below 70
KnowSafe	0	0	0
LSTM-RRT	2,958	45.3	15.9
Model-based	5,026	74.6	26.5

the model-based approach's limitation in predicting the system state only in the next time step, which proves insufficient to prevent or mitigate hazards under stealthy attacks.

Conversely, the ML-based approach LSTM-RRT reduces the maximum deviations to 45.3 mg/dL above 180 mg/dL and 15.9 mg/dL below 70 mg/dL, yet it still fails to mitigate 2,958 (45.2%) stealthy attacks due to unconstrained predictions.

In contrast, *KnowSafe* addresses these challenges by integrating both knowledge and data, enabling accurate prediction of system states for the subsequent two hours. This ensures sufficient reaction time and a high success rate in mitigating stealthy attacks.

3.4.5.8 Resource Utilization

We assess the resource utilization of the proposed approaches within the closed-loop simulation setting. By running simulations with various safety engines or without any mitigation/recovery 1,000 times, we calculate the average time overhead. The results depicted in Table 3.7 reveal that *KnowSafe* introduces an average overhead of 46.5 ms (PredNet) during normal operation, prior to entering mitigation mode, and 54 ms (SC-RRT* 25.3 ms, ActNet 28.7 ms) when in mitigation mode. These overheads are significantly smaller than the control cycle period in APS (5 minutes), ensuring minimal impact on system operation at runtime. Notably, *KnowSafe* does not introduce any time overhead after entering the mitigation mode when no updates to the corrective actions are made (see lines 9-13 in Algorithm 3).

Comparatively, the LSTM-RRT baseline exhibits a similar time overhead for hazard prediction and mitigation, albeit with a slightly larger overhead for the baseline RRT* algorithm (52.8 ms) due to processing more nodes resulting from looser constraints.

In contrast, the overall time overhead of the rule-based, model-based, and FFC baselines amounts to 121.5 μs, 87.9 ms, and 28.8 ms, respectively. This demonstrates that the overhead of *KnowSafe* is comparable to that of single models utilized in the model-based and FFC baselines.

Furthermore, we evaluate the resource utilization of *KnowSafe* on a typical MCU commonly used in APS (Raspberry Pi 4 with 8GB RAM as illustrated in Fig. 3.4), employing optimized ML implementation [136]. We observe a time overhead of 18 μs (4.12×10^{-7}% of the normal execution time of 4.36s) and 25.3 ms (5.8×10^{-5}%) before and after entering the mitigation mode, respectively. Additionally, the peak memory usage increases by 38MB (0.48% of the available 8GB RAM).

3.4.6 Evaluation of Robustness of ML-based Safety Monitors

Although robustness testing of deep learning models has been extensively explored in applications such as image classification and speech recognition, less attention has been paid to ML-driven safety monitoring in CPS. Therefore, in this research, we also evaluate the robustness of ML-based anomaly detection methods in safety-critical CPS against two types of accidental and malicious input perturbations in the input of ML models, which are different from the accidental faults or attacks that result in the unsafe control actions. We generate adversarial examples using a Gaussian-based noise model and the Fast Gradient Sign Method (FGSM) [115], since Gaussian is a reasonable assumption for any process or system that's subject to the Central Limit Theorem [137], and FGSM is a simple but effective method widely used in generating adversarial images, using the gradients of a neural network, and is reported to be also effective in non-image domain [138]. An example of how FGSM works is shown in Fig. 3.10 (Left).

Our experimental results with two case studies of APS for diabetes management show that an ML-based hazard prediction model trained with domain knowledge using a custom

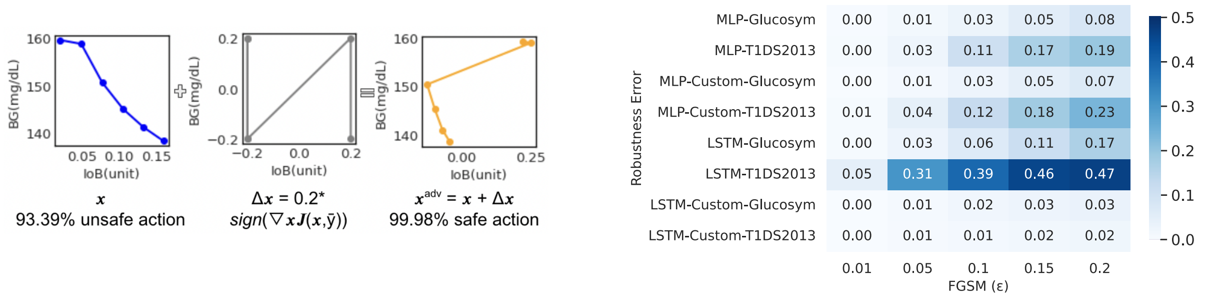


Figure 3.10: Left: An example FGSM attack on a baseline monitor with the *Keep_Insulin* injection command issued by the controller. Right: Robustness error of ML monitors against FGSM attacks.

loss function can reduce robustness error [3] by up to 54.2% on average (see Fig. 3.10 (Right)) and keep the average F1 scores high.

3.5 Discussion

Generalization and Limitation: We demonstrate the effectiveness of the proposed approach in two closed-loop APS testbeds and a real clinical dataset, but several limitations narrow the generalization of our approach. Specifically, our approach is limited to the CPS in which the system state can be represented by a subset of state variables, based on which the safety properties can be formally specified. Also, our approach is limited to CPS, where the control cycle and device memory are larger than this approach's requirement reported in Section 3.4.5.8.

The primary task of adapting our method to a new type of CPS involves creating safety specifications, which may not be easy for more complex systems and might require domain-specific technical expertise. This work adopts a formal framework that can generate formal safety specifications using a control-theoretic hazard analysis method [47,53] in cooperation with domain experts. An example of safety specification for Adaptive Cruise Control systems (ACC) for autonomous driving was explored in previous work [53]. Further study of the knowledge specification and hazard mitigation in different systems and scenarios is beyond the scope of the work.

Real-World Deployment: The proposed approach can be implemented on a smartphone (similar to some commercial APS as shown in Fig. 3.4) or be integrated with the pump microcontroller, but running ML models on resource-scarce embedded systems might be challenging. However, significant progress has been made in implementing ML models on embedded systems, such as using low-power embedded GPUs [139]), ML accelerators [140], or optimized ML implementations [136,141]. Further, some high-end MCUs have large memory and powerful computation capabilities [142]. In this work, we also implemented the proposed method on a typical MCU used in APS (Raspberry Pi 4 with 8GB RAM as shown in Fig. 3.4) with improved inference time (see Section 3.4.5.8). In addition, the integration of domain knowledge with the ML models is done offline during training without adding computational costs at run-time.

Our models are trained on patient/configuration-specific data. Changes in configuration, patient's physiological, or environmental context necessitate recalibration/retraining.

3.6 Related work

Anomaly Detection and Attack Recovery: In previous sections, we introduced existing works on anomaly detection and recovery using model-based, rule-based, or ML-based approaches. Our work distinguishes itself from these previous works in combining knowledge with data-driven techniques for preemptive hazard prediction and mitigation with better accuracy and a higher mitigation success rate. Further, this work focuses on mitigating or recovering the CPS from accidental faults or malicious attacks that directly compromise the controller software or hardware functionality in addition to detecting sensor attacks.

Security of APS: The existing works on enhancing APS security mainly focus on encryption [143], intrusion detection [144], safety specification [145], and model-based [146] or data-driven [47] run-time monitoring with less attention paid to hazard mitigation. The only work we found that attempts to avoid hazards in APS includes a fail-safe approach that shuts down the pump or stops insulin delivery under attack [126,127]. However, this approach fails to maintain the regular operation of APS. To the best of our knowledge, this work is the first on run-time hazard or attack mitigation in APS.

Knowledge Integration: Integrating expert knowledge into ML has been an active area of research [147]. Previous works have focused on enforcing ML to follow logic rules during the training process through applying soft constraints [131], designing specific logistic circuits [148] and network structures [149], generating graph models [150], or utilizing knowledge distillation [151–153] or logic-calibrated uncertainty [154]. In this work, we integrate domain-specific knowledge of safety constraints and properties as soft constraints with the multivariate sequential prediction models for anomaly detection and hazard mitigation.

3.7 Conclusion

This study introduces a novel approach that combines domain expertise with machine learning to develop safety engines capable of predicting and mitigating hazards in CPS. By integrating expert domain knowledge with machine learning techniques, we design custom loss functions to ensure adherence to domain-specific safety constraints during the training phase. Experimental findings from two closed-loop APS testbeds and a diabetic dataset demonstrate that the proposed safety engines exhibit enhanced accuracy in hazard prediction and a greater success rate in hazard mitigation.

Chapter 4

Context-Aware Safety Validation

4.1 Overview

Safety-critical CPS should be rigorously and thoroughly tested and validated before deployment. However, real-world testing is time and resource consuming and can be too risky during development. For example, it takes millions of hours of real road testing for the safety validation of a self-driving vehicle. Therefore, simulation-based validation is essential for assessing system resilience against unexpected events such as accidental faults, human errors, and attacks that might lead to adverse events. However, no realistic closed-loop testbed is found for some safety-critical CPS in the literature, such as the artificial pancreas systems (APS) or the autonomous driving system (ADS), with real-world control software and physical simulators as well as the model of real-world adverse events and existing safety interventions.

Another primary challenge in safety validation of CPS is to efficiently search the extensive large fault parameter space to find critical parameters that cause the system failures. Significant progress has been made in advancing safety validation techniques for CPS, such as sensitivity-based [8], model-based [9], or animation [156] methods. But with the increasing complexity of software-intensive CPS, there is a great need for more efficient and strategic validation techniques that are applicable to the systems at scale and unified methods that can rigorously trace from informal statements about a system and

This chapter contains material from the previously published works [111, 155], coauthored with H. Alemzadeh, M. Kouzel, A. Chen, H. Ren, M. McCarty, C. Nita-Rotaru, A. Schmedding, L. Yang, P. Schowitz, and E. Smirni, copyrighted by IEEE.

its (un)desired behavior to implemented code, hardware, physical behaviors, and their interactions with the environment. Previous works on safety validation using machine learning techniques have demonstrated improved efficiency in exploring the fault parameter space [49, 157–159]. However, the performance of these ML-based safety validation approaches heavily relies on the quality of the training data and requires a large amount of labeled data for both positive and negative classes, which is expensive to collect and might be unavailable in some applications (e.g., medical CPS). Further, the ML-based approaches are usually based on black-box models, which lack the transparency and interpretability essential for safety-critical applications. In addition, none of the previous works considered the existing safety checking mechanisms or the possibility of human interventions.

In this dissertation, we propose a context-aware safety validation strategy and optimization method together with a safety intervention simulator to explore key questions on how fault **timing** and **value** affect validation efficiency. Specifically, we propose a context-aware safety-critical validation strategy that can find the most critical context during a testing scenario to activate faults/attacks that strategically corrupt the control outputs or perception inputs, with the goal of (1) maximizing the chance of hazards and (2) causing hazards as soon as possible, before being detected or mitigated by the system operators or the existing safety mechanisms. We base this approach on high-level control-theoretic hazard analysis and specification of context-dependent safety requirements for a typical CPS (similar to the approach introduced in Chapter 2), which is applicable to any CPS with the same functional and safety specifications.

To bridge the sim-to-real gap in safety validation, we develop an open-source closed-loop experiment platform that integrates real-world control software and physical simulators together with typical safety mechanisms (e.g., AEBS) and a fault injection (FI) engine that simulates real-world adverse events in the literature.

4.2 Background

This work uses an advanced driver assistant system (ADAS), a level-2 ADS [160], as the primary case study. Fig. 4.1 shows the overall structure of a typical ADAS.

4.2.1 Advanced Driver Assistance Systems (ADAS)

Level-2 Advanced Driver Assistance Systems (ADAS) offer autonomous driving features but still require constant human attention [160]. Examples of these features include Adaptive Cruise Control (ACC), which manages the vehicle's longitudinal movement; Automatic Lane Centering (ALC), which handles lateral movement; and Advanced Emergency Braking System (AEBS), which operates braking through Automatic Emergency Braking (AEB) and delivers alerts via Forward Collision Warning (FCW). Currently, over 17 million passenger cars globally are equipped with ADAS [161].

The objective of ALC is to keep the vehicle centered within its lane, while the main goal of ACC is to maintain a safe following distance between the autonomous vehicle (also known as the Ego vehicle or AV) and the vehicle ahead in the same lane (known as the lead vehicle or LV). These goals are achieved by adjusting the Ego vehicle's driving direction and speed based on the detected lane lines and the estimated relative distance and speed to the lead vehicle.

Sensors. Existing DNN-based ADAS systems utilize various sensing technologies to detect lanelines and predict and track lead vehicles and objects. Systems like Tesla's Autopilot and Subaru's EyeSight rely solely on camera data, while others, such as Apollo [162] and OpenPilot, use both camera and radar data. Additionally, other sensors like GPS or IMU are employed to monitor the current speed and align it with the target speed set by human 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 lead vehicle

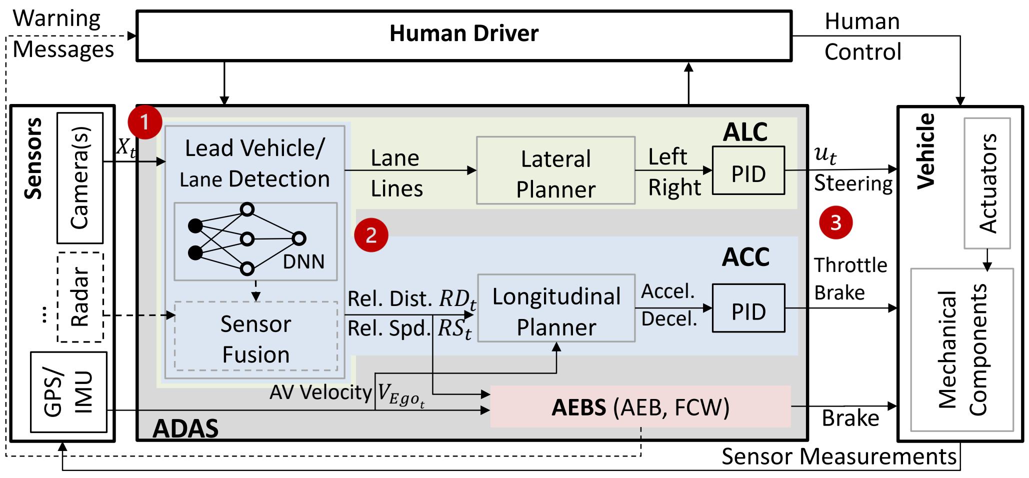


Figure 4.1: ADAS architecture with ACC, AEBS, and ALC.

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 [163] 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.

Lane Detection. The critical component of ALC is laneline detection, which detects the lanelines/boundaries of driving lanes using camera data or a fusion camera and Lidar data.

Planner. The next stage in the process involves determining the optimal driving direction and speed based on the outputs from lane detection and LVD, as well as the current state of the vehicle. The lateral and longitudinal planners employ algorithms such as Model Predictive Control (MPC) to generate multiple desired direction and speed trajectories. Each trajectory represents a series of directions and speeds over a specified following period [52].

Vehicle Control. At each control cycle t, the ADAS selects the plan from the lateral and longitudinal planners that offers the lowest speed and risk (such as the risk of colliding with the lead vehicle or veering out of the lane). This plan is then input into Proportional-Integral-Derivative (PID) controllers [51], which compute the specific

Table 4.1: AEBS design in OpenPilot-supported car models.

Car Model [84]	**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

optimal control command u_t. This command determines the appropriate steering angle and throttle or brake amount, ensuring that the vehicle accurately and promptly follows the desired direction and 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}.

4.2.2 ADAS Safety Mechanisms

The Advanced Emergency Braking System (AEBS), including Forward Collision Warning (FCW) and Automatic Emergency Braking (AEB), is a fundamental safety mechanism designed to alert drivers about potential collision risks with an obstacle ahead and actively decelerate the vehicle to prevent accidents. Current Advanced Driver Assistance Systems (ADAS) predominantly incorporate AEBS under various names. As shown in Fig. 4.1, most AEBS implementations utilize both camera and radar for collision prediction through sensor fusion [164] and make control actions based on LVD outputs and other sensor measurements (see Table 4.1). Additionally, safety principles such as maximum acceleration limits required by international standards (e.g., ISO 22179) and firmware safety checks (e.g., constraints on the output steering angle) are incorporated into the design of typical production ADAS to ensure driving safety [165].

Table 4.2: Attacks comparison for DNN perception or ADAS.

Attack	**Attack**			**Safety Interventions**		**Autonomy**
Method	**Type**	**Vector**	**Target**	**AEBS**	**Driver**	**Level** [160]
[166]	Offline	Stickers on road signs	Classifier	N	N	N/A
[167]		Patch projected	MTO	N	N	L4
[168]		Stickers on road	ALC	N	N	L2
[169]		Patch on truck	ACC	N	N	L2
[170]		Patch on road	ALC	**Y**	N	L2
[171]	Runtime	Perception inputs	MTO	N	N	L4
[157]		Perception inputs	MTO	No FCW	N	L4
[172]		Inner state variables	FCW	No AEB	Y	N/A
[111]		Control commands	ACC, ALC	No AEB	**Y**	L2
Ours		Perception inputs	ACC	**Y**	**Y**	L2

* MTO: Multi-Object Tracking;

Previous studies on the safety of autonomous driving tend to focus on Level 4 or fully autonomous vehicles, often neglecting the impact of human driver interventions during emergency situations (e.g., abnormal acceleration). Additionally, some research has overlooked the inclusion of basic safety features like AEB or FCW and their impact on the effectiveness of attacks (see Table 4.2). For a realistic assessment of ADAS security, it is essential to evaluate how these safety features interact with human interventions. At Level 2 automation, drivers must maintain control and supervise ADAS functionalities [160]. There is a research gap regarding how to ensure the safety of the combined operation of human drivers and autonomous vehicles. The primary challenge lies in assessing the ability of human drivers to anticipate and respond to situations where automation may fail.

Time-to-Hazard (TTH). Previous research has indicated that there is generally a lag between the initiation of faults and the issuance of unsafe control commands to the physical layer, leading to hazardous situations [38, 47]. This interval, from the start of an attack to the manifestation of a hazard, is termed the Time-to-Hazard (TTH). The TTH metric represents the critical period available for detecting anomalies and implementing mitigation measures, as depicted in Fig. 4.2.

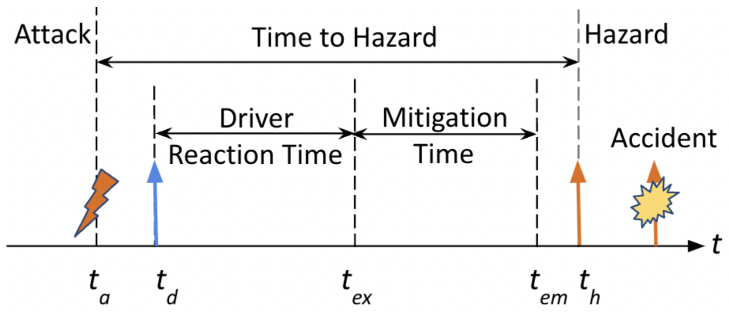


Figure 4.2: Timeline of attack propagation. (t_a: Attack activated; t_d: Attack is detected by the ADAS or the anomaly is sensed by the human driver; t_{ex}: Human driver starts to engage; t_{em}: End of mitigation; t_h: Hazard occurs.)

Driver Reaction Time is defined as the time difference between the perception of an alert or anomaly (e.g., seeing an alert raised by the ADAS or recognizing an anomaly) and the start of physically taking an action (e.g., hitting the brake). In the AV literature, the overall driver reaction time (perception and reaction) is reported to be 2.5 seconds on average [170,173]. We define the *Mitigation Time* as the time it takes for any corrective actions (e.g., braking) to be completed. This timing provides a window of opportunity for attackers to cause hazards before being overruled by the human driver or automated safety mechanisms. Fig. 4.2 shows an example where mitigation successfully completed before the occurrence of the hazard ($t_{em} < t_h$). A successful attack should evade detection and/or lead to hazards before the ADAS or the driver engage ($t_h < t_{ex}$) or complete any mitigation actions ($t_{ex} < t_h < t_{em}$).

4.2.3 OpenPilot

We utilize OpenPilot, a production ADAS from Comma.ai, as our case study [83]. OpenPilot is unique as it is the only open-source Level-2 ADAS, designed to enhance visual perception and automated control (featuring ACC and ALC) by connecting an additional EON device to the vehicle's OBD-II port. The ACC system in OpenPilot follows a typical deep neural network (DNN)-based architecture, as illustrated in Fig. 4.1, with an end-to-end system design [174].

Currently, OpenPilot supports over 250 car models, including brands like Toyota and Honda [84]. It boasts over 10,000 active users and has accumulated more than 100 million miles of driving on public roads [83]. According to Consumer Reports, OpenPilot achieves state-of-the-art autonomous driving performance, outperforming 17 other production ADAS on the market, including Tesla Autopilot and Audi Driver Assistance Plus [175].

The DNN model used by OpenPilot, called Supercombo, utilizes an EfficientNet-B2 based CNN model to process image data [176]. 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, lead vehicles, and vehicle pose, resulting in a total of 6,472 outputs. For more details on the specifications of Supercombo, refer to the link [1].

Comma.ai provides a closed-loop simulation environment that integrates OpenPilot with CARLA, a high-fidelity urban driving simulator capable of generating near-realistic camera image frames [177]. However, in this default simulation setup (described in Section 2.8.1.2), sensor fusion relies solely on camera data due to the absence of radar sensors. Additionally, typical ADAS safety mechanisms are not included.

4.3 Attack Model

Attacker Objective. The objective of the attacker is to induce collisions or cause the vehicle to drive out of its lane,while remaining stealthy to avoid being detected or prevented by the human driver or safety mechanisms (e.g., AEB, FCW), by targeting the DNN inputs (❶ in Fig. 4.1) or directly manipulating the DNN outputs ❷ or control outputs ❸, depending on their capabilities and access level to the ACC system.

[1]https://github.com/commaai/openpilot/tree/90af436a121164a51da9fa48d093c29f738adf6a/selfdrive/modeld/models

Attacker Knowledge. We assume that the attacker gains comprehensive knowledge about the target ADAS system design and implementation by reverse engineering a purchased or rented vehicle with identical control software as the victim vehicle [5,168] or by studying publicly available documents or source code. This is possible given that some of the production ADAS systems are open source [83,162].

Attacker Capabilities. We assume the attacker has the capability to intercept sensor measurements and change perception inputs or control outputs at runtime.

A possible way to achieve this is to implant malware by compromising the over-the-air (OTA) update mechanisms [178–181] or gaining one-time remote access to the ADAS software through scanning the network, accessing stolen credentials and exploiting the vulnerabilities in SSH protocol [182], browsers, access control [183], wireless communications [183–185], third-party components connected to in-vehicular network [186], or some remote service/backdoor offered by the manufacturer (e.g., Comma Connect for OpenPilot [187] 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 [188]. 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 [181,189], 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 perception 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 [190], CAN Bus [179,183,192,197] or Ethernet channel [157,191]), to read and send image data at runtime. Additionally, this approach can be used to alter control outputs, which are typically communicated via the CAN Bus. The attacker computes the attack value on a local wireless device or a remote server.

Table 4.3: Threat models: attacker strength, capability, and impact.

Threat Model	**Attacker Strength**	**Access to ADAS Software**	**Vehicular Networks**		**Computation Location**	**Impact**	**Examples**
			Read	**Write**			
Malware	Strong[1]	✓	✓	✓	within ADAS	Fleet of Vehicles	[181, 189]
Wireless	Medium[2]		✓	✓	Local Device, Remote Server	Single Vehicle	[190] [191] [157] [183] [192]
Physical	Weak[3]		✓		Remote Server	Single Vehicle	[193] [194] [195] [196] [167]

[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

Further, physical attack methods are also viable for perception attack, such as by displaying the patch on a monitor attached on the rear side of a leading adversarial vehicle [193,194] or projecting the patch into the rear of the lead vehicle using a projector [167,195,196].

Table 4.3 summarizes various methods for runtime reading of perception data and modifying of live camera frames or controller outputs, given different attacker strengths and capabilities. In this dissertation, we mainly focus on the runtime and optimized modification of live camera frames or control outputs to enhance the attack success rate and stealthiness, regardless of the exact threat model and how the attacker obtained access. In our experiments, we implemented the attack through malicious OTA update to OpenPilot.

4.4 Attack Challenges

Several challenges need to be addressed in attacking DNN-based ADAS 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 [49,75,111] 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 lead vehicle is detected. Recent works have focused on using machine learning to explore the fault/attack parameter space [198] and improve the attack effectivenes [49, 157], 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 ADAS 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 [166, 169] are inadequate for runtime perception attacks 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 lead vehicle'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 [166] or a known set [169, 170] with unlimited computation resources, allowing iterative optimization until an optimal attack vector is generated. However, attacking ADAS 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 ADAS systems (e.g., frame rate of 20Hz and control cycle of 10 milliseconds in [83]) significantly impact the effectiveness of optimization-based attack strategies.

4.5 Context-Aware Attack Activation

To determine the most critical times for activating the attack or conducting safety validation (**C1**), we employ a control-theoretic hazard analysis method [14, 199] and follow the framework outlined in Section 2.3 to identify the most critical system contexts. These contexts pertain to specific control actions that may lead to hazards if issued by the CPS control software. This approach primarily relies on domain knowledge concerning system safety requirements and does not necessitate large amounts of training data or computational resources, unlike an ML-based approach.

Taking an ADAS as an example, the attacker's goal is to manipulate perception inputs or control commands such as gas, brake, and steering angle to increase the likelihood of hazardous events while evading detection by the ADAS safety mechanisms and the human driver. Specifically, the attacker aims to cause the following accidents:

- **A1:** Collision with the lead vehicle.
- **A2:** Rear-end collision, resulting in traffic congestion.
- **A3:** Collision with road-side objects or other vehicles in the neighboring lane.

These accidents could happen by forcing the system to transition into one of the following hazardous states:

- **H1:** AV violates safe following-distance constraints with the lead vehicle, which may result in A1.
- **H2:** AV decelerates to a complete stop although there is no lead vehicle, which may lead to A2.
- **H3:** AV drives out of lane, which may lead to A3.

Table 4.4 illustrates an example context table detailing unsafe system contexts, delineating specific high-level system conditions wherein certain types of control actions might

Table 4.4: Safety context table for an ADAS with ALC and ACC

Rule	System Context	Control Action	Potential Hazard
1	$HWT \leqslant t_{safe} \wedge RS > 0$	u_1	H1
2	$HWT > t_{safe} \wedge RS \leqslant 0 \wedge Speed > \beta_1$	u_2	H2
3	$d_{left} \leqslant 0.1m \wedge Speed > \beta_2$	u_3	H3
4	$d_{right} \leqslant 0.1m \wedge Speed > \beta_2$	u_4	H3

* HWT: Headway Time = Relative Distance/Current Speed;
* RS: Relative Speed = Current Speed - Lead Speed;
* d_{left}, d_{right}: Distance to the left/right edge of current lane;
* $u_{1,2,3,4}$: Acceleration, Deceleration, Steering Left, Steering Right.
* $t_{safe} \in [2, 3]s$, $\beta_1, \beta_2 \in [20, 35]mph$

engender safety hazards. For instance, the first row specifies that if the Headway Time falls below a safety threshold t_{safe} (e.g., 2 seconds), and the autonomous vehicle's (AV) speed exceeds that of the lead vehicle ($RS > 0$), executing an acceleration control action becomes perilous as it could culminate in a collision with the lead vehicle. This identification of context-specific unsafe control actions can be undertaken by an attacker who possesses knowledge regarding the typical functionalities of an ADAS and can be applied universally to any ADAS with an identical functional specification. The precise values for the unknown thresholds t_{safe}, β_1, and β_2 can be determined based on domain expertise or historical data [47].

4.6 Strategic Value Corruption

4.6.1 Attacking Control Output

Our proposed attack strategy uses the critical system contexts described in Table 4.4 as the trigger for injecting unsafe control commands [200]. To evade detection, the control actions generated by the attack must be within the limits that are not noticeable to a human operator and are checked by the ADAS safety mechanisms, while minimizing the Time to Hazard (TTH) (see Fig. 4.2) and maximizing the chance of resulting in any hazards. To achieve these goals, the following optimization problem is formulated:

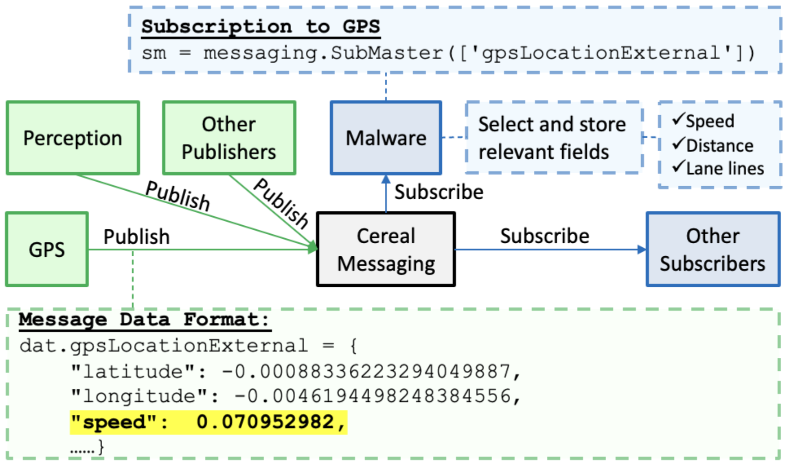
$$\begin{aligned} & minimize_{TTH} max\{Pr\{x_{t_{+TTH}} \in Hazardous\}\} && (4.1) \\ & s.t.\ brake \geq limit_{brake} \\ & \quad accel \leqslant limit_{accel} \\ & \quad \Delta steering < limit_{steer} \\ & \quad \hat{v}_{t+1} \leqslant 1.1 v_{cruise} \\ & \quad \hat{v}_{t+1|t} = \hat{v}_t + accel * \Delta t && (4.2) \\ & \quad \hat{v}_{t+1} = \hat{v}_{t+1|t} + K_t * (v_{t+1} - \hat{v}_{t+1|t}) && (4.3) \end{aligned}$$

where *brake*, *accel*, and *steering* indicate the modified values of control commands, and $\hat{v}_{t+1}$ represents the predicted speed of the Ego vehicle at the next time step, which can be estimated using Eq. 4.2 that approximates the dynamics of the vehicle by assuming linear acceleration for a short time period Δt (10 ms). A Kalman filter [163] (with the Kalman Gain parameter K_t, see Eq. 4.3) is used to update the estimation using the measured speed v_{t+1} at the next time step. $limit_{accel}$, $limit_{brake}$, $limit_{steer}$ are the constraints on the output control commands defined by the safety checking rules of the target vehicle, including those of its ADAS.

4.6.1.1 Attack Procedure

The overall procedure and steps for executing Context-Aware attacks are summarized as follows:

Eavesdropping: This step involves monitoring the sensor sockets and the in-vehicle communication network, decoding messages exchanged between different software components, and extracting sensor data and critical state information. In OpenPilot, this can be achieved by subscribing locally or remotely to the messaging system used for internal packet communication, known as Cereal [201]. Cereal is a publisher-subscriber messaging specification for robotic systems, similar to ROS [202]. It is used by sensing and per-


```
Subscription to GPS
sm = messaging.SubMaster(['gpsLocationExternal'])
```

```
Message Data Format:
dat.gpsLocationExternal = {
    "latitude": -0.00088336223294049887,
    "longitude": -0.0046194498248384556,
    "speed":  0.070952982,
    ......}
```

Figure 4.3: Cereal messaging eavesdropping.

ception modules (e.g., GPS, Radar) to publish messages, which can then be subscribed to by other OpenPilot modules (e.g., ACC, ALC) as well as any malicious software (see Fig. 4.3).

Since OpenPilot is open-source, the format of cereal messages is publicly available [203]. An example of eavesdropping on the GPS messages is shown in Fig. 4.3. To extract the information needed for safety context inference, the attacker needs to subscribe to the following events: 1) "*gpsLocationExternal*" events to learn the speed of the Ego vehicle published by GPS; 2) "*modelV2*" events to receive messages from the perception module to learn the lane line positions; 3) "*radarState*" events published by the RADAR to learn the relative speed and distance of the lead vehicle.

Safety Context Inference: Next, the attacker utilizes the basic state information x_t, which includes the speed of the Ego vehicle, the vehicle's lateral position, the lane line positions, and the relative distance to the lead vehicle. This data is used to infer more complex and human-interpretable state variables as outlined in the safety specification (Table 4.4). For instance, headway time (HWT) is a crucial metric for identifying critical system contexts and can be computed using the Ego vehicle's current speed and its relative distance to the lead vehicle.

Attack Type and Activation Time Selection: A context matcher identifies

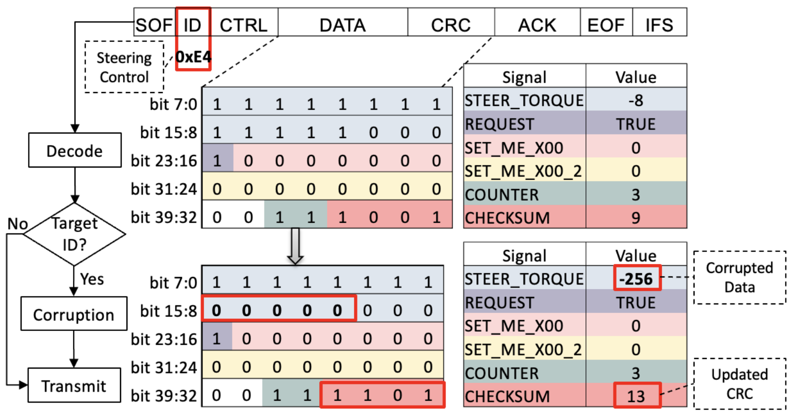


Figure 4.4: An example of changing a steering output CAN message.

whether the current system state aligns with any critical system contexts specified in Table 4.4. Upon finding a match, the attack engine determines the appropriate attack action (e.g., *Acceleration*) based on the specified unsafe action for that context and initiates the attack. Table 4.5 (in Section 4.8.1) outlines the types of attacks to be triggered for various contexts. If two different context conditions are detected simultaneously, both control actions (e.g., *Acceleration* and *Steering*) are activated.

Strategic Value Corruption: In the final step, the chosen attack type (e.g., *Acceleration*) is converted into low-level control commands (e.g., maximum *gas* and zero *brake*). The attack engine dynamically corrupts these control command values while ensuring they do not exceed the safety limits enforced by OpenPilot's safety mechanisms (refer to Eq. 1-3). These safety limits are identified and encoded offline using information from open-source code and publicly available documentation.

Finally, the corrupted commands are transmitted to the target actuators by manipulating CAN messages. The data in a CAN bus message can be decoded using reverse engineering and the open-source Database Container (DBC) configuration [204, 205] specific to a car model. The attack engine then corrupts the particular CAN message containing the target control command by using the command's unique identifier (e.g., 0xE4 for steering, as illustrated in Fig. 4.4). After corrupting the target control commands, the attacker updates the checksum to maintain the integrity of the corrupted CAN message.

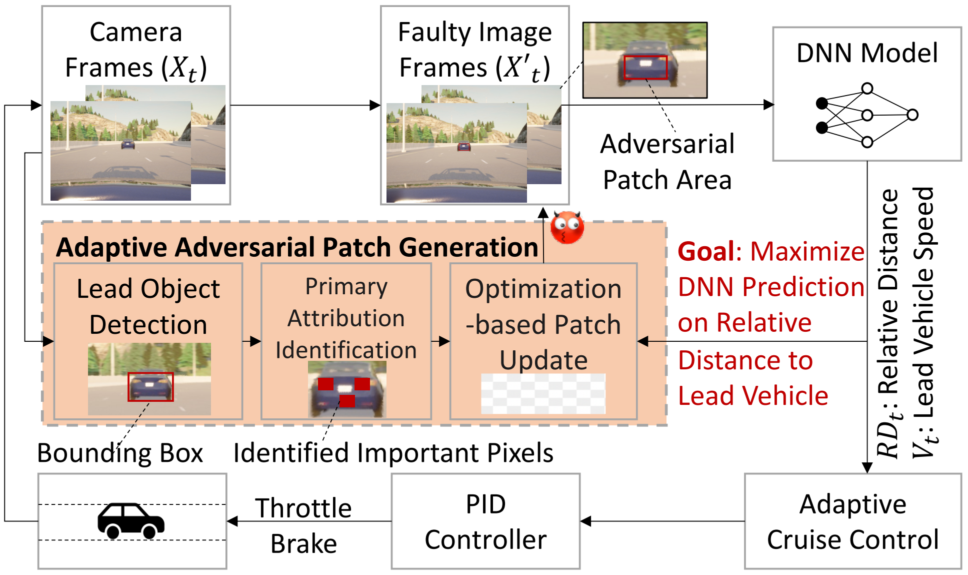

Camera Frames (X_t)
Faulty Image Frames (X'_t)
Adversarial Patch Area
DNN Model
Adaptive Adversarial Patch Generation
Lead Object Detection
Primary Attribution Identification
Optimization -based Patch Update
Goal: Maximize DNN Prediction on Relative Distance to Lead Vehicle
RD_t: Relative Distance
V_t: Lead Vehicle Speed
Bounding Box
Identified Important Pixels
Throttle
Brake
PID Controller
Adaptive Cruise Control

Figure 4.5: Optimization-based adversarial patch generation.

4.6.2 Attacking Perception Input

The critical system contexts identified in Section 4.5 are based on the high-level unsafe actions (e.g., Acceleration) issued by the ADAS 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. 4.5.

4.6.2.1 Runtime Optimization-based Adversarial Patch Generation.

To address challenge **C2**, we formulate the attack as the following runtime optimization problem:

$$\min \sum_{d \in RD_t} -\nabla g(d,\theta) + \lambda ||\Delta_t||_p \tag{4.4}$$

$$s.t.\ \ Patch_t = \Delta_t * M_t \tag{4.5}$$

$$Patch_t \in [\mu - \sigma, \mu + \sigma] \tag{4.6}$$

$$Area(Patch_t) \subset BBox(LV)_t \tag{4.7}$$

$$X_t^{adv} = X_t + Patch_t \tag{4.8}$$

$$[RD, RS]_t = LVD_\theta(X_{t-1}^{adv}) \tag{4.9}$$

$$u_t = ACC(s_t, [RD, RS]_t) \tag{4.10}$$

$$s_{t+1} = CarModel(s_t, u_t) \tag{4.11}$$

where Eq. 4.4 defines an objective function that aims to accelerate the Ego vehicle as quickly as possible to cause a forward collision. Directly reducing the probability of detecting a lead vehicle or its bounding box (BBox) does not alter the behavior of the ACC system. Instead, we design an objective function that maximizes RD_t while ensuring the perturbation applied to the adversarial patch remains imperceptible to human eye.

In Eq. 4.4, $g(d)$ 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 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 4.3).

In Eq. 4.8, 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

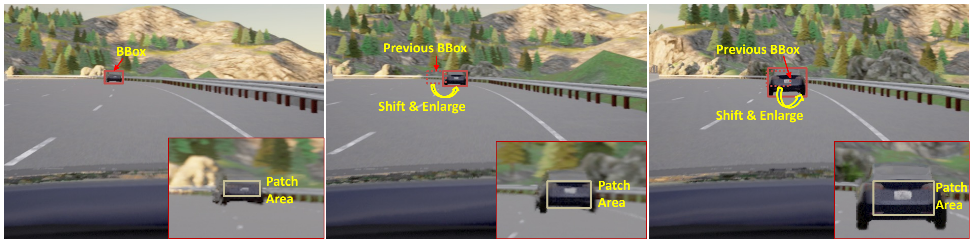


Figure 4.6: Examples of the shift and adjustment process in the patch generation due to the changes in the lead vehicle's position and size from the driver's view. Inset figures are the zoomed-in views of the front vehicle with an adversarial patch added around the license plate area.

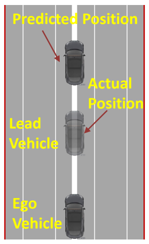


Figure 4.7: ACC under attack.

stealthiness. We limit the perturbation value within the Kalman filter noise parameters (μ,σ) (Eq. 4.6), which ensures the perturbation is not corrected by the sensor fusion. We also constrain the adversarial patch inside the BBox of the lead vehicle (Eq. 4.7) to enhance attack effectiveness, minimize the perturbation area for stealthiness, and reduce computational cost.

4.6.2.2 Primary Attribution Detection and Patch Update.

As discussed in Section 4.4, a significant challenge (C3) in designing runtime attacks is the variability in the size and location of the lead vehicle within the perceived image frames. To address this issue, the attacker must dynamically update the generated patch based on the approximate DNN outputs. In this work, we employ an object detection method [206] to detect and track the real-time position and dimensions of the lead vehicle, focusing the attack perturbation within the detected bounding box (BBox) of the lead vehicle. In production ACC systems with integrated object detection features [162], and given appropriate access, the attack can bypass this step and directly utilize the stock prediction results.

After obtaining the BBox, we employ a primary attribution algorithm [207] to quantify the relationship between input features and output predictions. This process helps us identify the key pixels within the BBox of the lead vehicle that significantly contribute to the predictions of RD_t. The input pixels with high weights, as determined by the attribution algorithm, are marked with a unit value in the mask matrix M_t, while the remaining pixels are assigned zero values. This mask matrix is then multiplied by the perturbation Δ_t to generate the adversarial $Patch_t$ (as described in Eq. 4.5). This step is beneficial as it filters out non-important pixels, reducing the number of perturbed pixels to improve the efficiency of optimization-based attacks and lower the computational 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. 4.12-4.14). We shift the attack vector toward the new position of the detected BBox of the lead vehicle 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 the BBox at the previous and current control cycles, respectively (Eq. 4.12). We then expand the adversarial patch attack vector ($Patch$) to the dimensions that match the size of the newly detected BBox of the lead vehicle. Instead of reinitializing the entire attack vector matrix with random or zero values, which would reset the whole optimization process, we retain the previous patch values and intermediate variables, initializing only the newly expanded units (Eq. 4.13-4.14). An example is shown in Fig. 4.6.

$$Pos(Patch_t) = Pos(Patch_{t-1}) + (x_t - x_{t-1}, y_t - y_{t-1}) \tag{4.12}$$

$$Init(\Delta_t) = [0] * size(BBox(LV)_t) + \begin{bmatrix} \Delta_{t-1} & 0 \\ 0 & 0 \end{bmatrix} \tag{4.13}$$

$$Init(Patch_t) = Init(\Delta_t) * M \tag{4.14}$$

This algorithm maintains a continuous optimization process across two consecutive perception cycles, which is critical in satisfying real-time constraints. Fig. 4.7 shows a visualization of how the adversarial patch affects the DNN predictions.

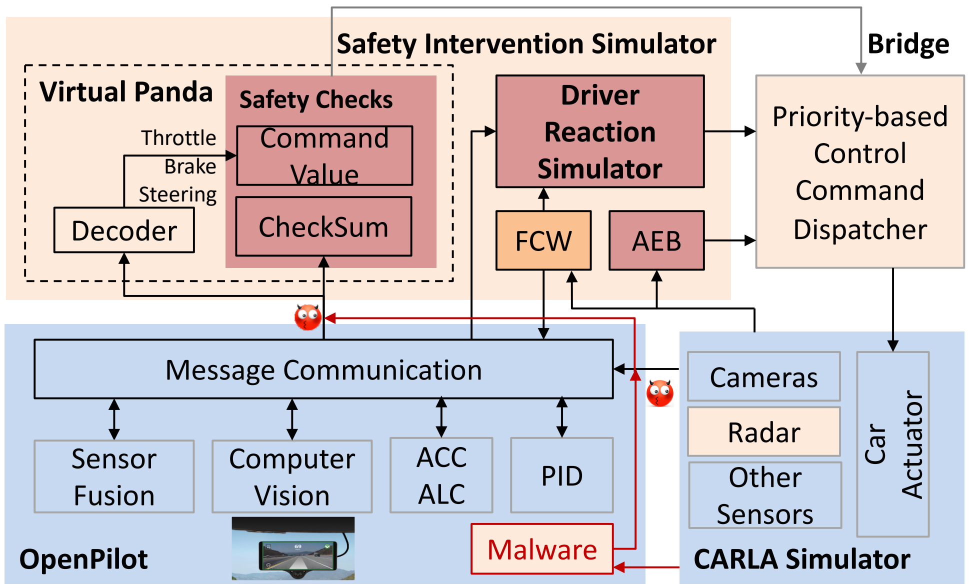


Figure 4.8: Simulation platform: OpenPilot integrated with CARLA simulator and safety interventions, including driver reaction simulator, virtual Panda safety constraints checking, FCW, and AEB.

4.7 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 2.8.1.2 and Section 4.2.3) 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 (see Appendix A.2.3) and fusion of camera and radar data (see Appendix A.2.4). An overview of the simulation platform is shown in Fig. 4.8 (with the orange parts representing our new implementations) and presented next.

To bridge the gap in accounting for safety interventions and address the challenge of ensuring the combination of human driver and vehicle safe (see Section 4.2.2), we have implemented and integrated three levels of safety interventions into the OpenPilot software (see Fig. 4.8). These include ADAS safety features (e.g., AEB and FCW), basic vehicle safety constraint checks on control commands, and driver interventions.

AEBS (FCW and AEB) Simulator. For designing and testing the AEBS mechanisms in simulation, we thoroughly review the regulations and requirements concerning AEBS [208–210] and adhere to UN Regulation No. 152 [209]. We adopt and implement a time-to-collision (TTC) based phase-controlled AEBS [211] in our platform.

The AEBS processes inputs derived from lead vehicle detection (LVD) outputs after sensor fusion, including relative distance (RD), relative speed (RS), and the current speed of the Ego vehicle (V_{Ego}) (see Fig. 4.1). The average driver reaction time (T_{react}) is standardized to 2.5 seconds, a commonly accepted value in the literature [111, 170]. Various time thresholds are then computed, including ttc (time to collision), t_{fcw} (forward collision warning time), t_{pb1} (first phase partial brake time), t_{pb2} (second phase partial brake time), and t_{fb} (full brake time). 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 value blocks other controls from the ADAS. More details of AEBS design and testing are provided in Appendix A.2.1.

In practice, when OpenPilot is installed on different car models, some may lose AEBS functionality [84], while others retain it. Additionally, AEBS might depend on a separate ADAS camera [212], distinct from the OpenPilot camera, which is also vulnerable to potential data compromise. Therefore, 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 4.8.6.2 and Table 4.10).

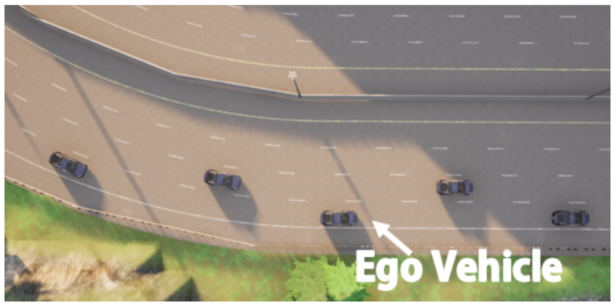
Driver Reaction Simulator. To evaluate driver interventions, we develop a driver reaction simulator. The simulated driver receives notifications when any ADAS safety alerts are triggered (e.g., FCW) or when the driver notices anomalies in the vehicle's status or camera user interface (UI) (e.g., the mean perturbation value in the UI exceeding a noticeable threshold, set by default to 15% for an alert driver ($Patch.mean() > 0.15$). Assuming a highly alert driver capable of detecting anomalies within a single control cycle (10ms), the driver issues a predefined emergency response that takes effect 2.5 seconds later, which is the average driver reaction time. Refer to Appendix A.2.2 for more details about the design of the driver reaction simulator.

Safety Constraint Checker. The OpenPilot safety mechanisms are implemented in its control software and the Panda CAN interface. Panda, a universal OBD adapter developed by Comma.ai [213], provides access to almost all car sensors through the CAN bus and enforces 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.

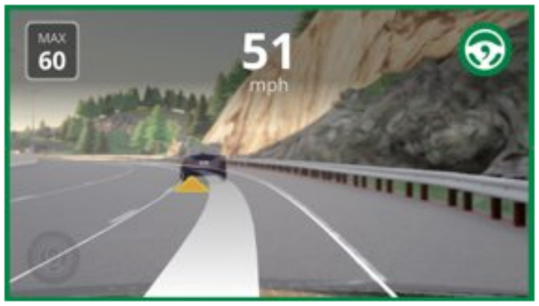
To ensure our simulation is as realistic as the actual OpenPilot on the road, we add a *virtual Panda* module that replicates the exact logic of the Panda software [213]. Specifically, as shown in Fig. 4.8, the virtual Panda decodes the CAN messages sent by the ADAS and verifies their checksums and control command values against predefined thresholds [213]. For example, to ensure safety, the maximum acceleration and deceleration of the vehicle are limited to $2m/s^2$ and -3.5m/s^2, respectively [165]. Only commands that pass the Panda safety checks are sent to the simulated vehicle actuators. In the CARLA simulator, the final control commands are truncated within the range of [0,1].

4.8 Evaluation in Closed-loop Simulation

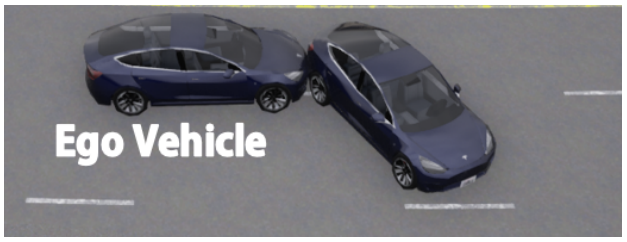
We evaluated the performance of the proposed approaches using the developed open-source closed-loop simulation platform, as described in Section 4.7 and illustrated in Fig. 4.8. Our experiments were conducted on Ubuntu 20.04 LTS, utilizing OpenPilot v0.8.9 and CARLA v9.11. Each simulation of OpenPilot consists of 5,000 time-steps, each lasting approximately 10 ms, resulting in a total duration of 50 seconds. However, if an attack results in a collision, the simulation terminates earlier.



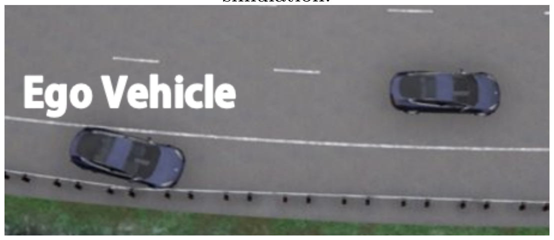
(a) An example initial position of Ego Vehicle (EV) and other reference vehicles.



(b) The user interface of OpenPilot during the simulation.



(c) EV collides with the lead vehicle.



(d) EV collides with the guardrail.

Figure 4.9: Driving scenarios in OpenPilot.

4.8.1 Driving Scenarios

We simulated a 2016 Honda Civic, both with and without basic safety features, navigating curvy and straight sections of a highway on the "Town04_opt" map in CARLA, under clear weather and dry road conditions. Our simulations included four high-risk driving scenarios, designed in accordance with the NHTSA's pre-collision scenario topology report [87]. In these scenarios, the Ego vehicle, traveling at 60 mph, encounters a lead vehicle that exhibits a range of behaviors:

- SC1: Lead vehicle cruises at the speed of 35 mph;
- SC2: Lead vehicle cruises at the speed of 50 mph;
- SC3: Lead vehicle slows down from an initial speed of 50 mph to 35 mph;
- SC4: Lead vehicle accelerates from an initial speed of 35 mph to 50 mph.

Fig. 4.9(a-b) show different views of a simulated scenario.

Table 4.5: Fault injection experiments.

Attack Location	**Attack Type**	**Accel**	**Brake**	**Steering Angle**	**No. Attacks**
Control Outputs	Acceleration	$limit_{accel}$	0	-	60
	Deceleration	0	$limit_{brake}$	-	60
	Steering-Left	-	-	-$limit_{steer}$	60
	Steering-Right	-	-	$limit_{steer}$	60
	Acceleration-Steering	$limit_{accel}$	0	$\pm limit_{steer}$	60
	Deceleration-Steering	0	$limit_{brake}$	$\pm limit_{steer}$	60
Perception Inputs	Adversarial Patch	-	-	-	250

For each driving scenario, we simulate various types of attacks by adding adversarial patches to the perception inputs or injecting faults into each output variable and their combinations, as outlined in Table 4.5. These modifications can induce sudden accelerations in the AV. For instance, in the Acceleration-Steering attack, faults are injected into the Gas and Steering Angle parameters, either to the left or right angle, within limits acceptable to the OpenPilot control software or as specified in Section 4.6.

For control output attacks, we test each scenario with the lead vehicle starting at three different distances (50m, 70m, 100m) and repeat each test 20 times to account for variations in the driving environment and attack timing. This approach results in 60 simulations per attack type and a total of 1,440 simulations across all attacks and scenarios.

For perception attacks, the AV starts 75 meters away from the lead vehicle. Each scenario is tested across ten start times and five durations, repeated five times, leading to a total of 1,000 simulations. This extensive testing helps ensure a robust evaluation of the attack impacts under various conditions.

4.8.2 Methodology

We study the following research questions by comparing the effectiveness of the proposed perception attack (referred to as **CA-Opt**) and output attack (referred to as **Context-**

Table 4.6: Overview of attack strategies.

Attack Location	**Attack** Strategy	**Start** Time	**Duration**	**Attack** Values	**No.** Attacks
Control Outputs	Random-ST+DUR	Uniform [5,40]s	Uniform [0.5,2.5]s	Fixed[1]	14,400
	Random-ST	Uniform [5,40]s	2.5s	Fixed	1,440
	Random-DUR	Context-Aware	Uniform [0.5,2.5]s	Fixed	1,440
	Context-Aware (Ours)	Context-Aware	Context-Aware	Strategic[2]	1,440
Perception Inputs	CA-Random	Context-Aware	Context-Aware	Random	1,000
	CA-APGD	Context-Aware	Context-Aware	AutoPGD	1,000
	CA-Opt (Ours)	Context-Aware	Context-Aware	Opt-based	1,000

[1] Fixed: use the maximum limit of each output command defined in OpenPilot: $limit_{steer} = 0.5°$, $limit_{brake} = -4m/s^2$, $limit_{accel} = 2.4m/s^2$.
[2] Strategic: dynamically choose the attack value according to Eq. 4.1-4.3 ($limit_{steer} = 0.25°$, $limit_{brake} = -3.5m/s^2$, $limit_{accel} = 2m/s^2$).

Aware) to several baseline attack methods in causing safety hazards and evading different safety interventions:

RQ1: Does strategically selecting attack times and values increase the chance of hazards, such as 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 various baseline attack strategies to answer these questions (see Table 4.6).

To evaluate the resilience of ADAS against control output attacks and assess the influence of attack timing, we developed three baseline strategies in addition to the Context-Aware strategy, as detailed in Table 4.6. The first baseline, **Random-ST+DUR**, uses randomly selected start times, uniformly distributed between 5 and 40 seconds after the simulation starts and up to 10 seconds before it ends, with attack durations uniformly distributed between 0.5 and 2.5 seconds. We conducted 14,400 simulations using the Random-ST+DUR strategy to comprehensively test critical attack parameters. The second baseline, **Random-ST**, involves randomly selecting a start time while fixing the attack duration at the average driver reaction time of 2.5 seconds. The third baseline, **Random-DUR**, randomly selects the attack duration between 0.5 to 2.5 seconds, with the start time determined based on the context.

All attack values are kept within the safety parameters monitored by OpenPilot. We exclude aggressive random attacks, such as those bombarding the CAN-bus with out-of-range values, as they might be detected by existing vehicular network intrusion detection systems [5,214] and OpenPilot's safety checks.

To evaluate the effectiveness of our optimization-based adversarial patch method in strategically selecting attack values, we compared it with two baselines: **CA-Random**, which introduces random perturbations to perception inputs, and **CA-APGD**, a state-of-the-art gradient-based method using Auto-PGD [215]. Originally designed for misclassification, Auto-PGD is not suited for ACC attacks; therefore, we modified its goal function to maximize the relative distance prediction (see Eq. 4.4). Additionally, we restricted the number of iterations to 5 to align with the maximum number of control cycles (100Hz) within a perception cycle (20Hz). Both baselines employ the same context-aware method as our proposed CA-Opt attack to select the start times and durations of attacks. For a balanced comparison, we confined the perturbations to the detected bounding box (BBox) of the lead vehicle. Furthermore, we continuously updated the BBox size and position using our proposed patch updating algorithm (Section 4.6.2.2). This ensures that all comparisons are conducted under similar operational conditions.

4.8.3 System Resilience Evaluation

We assess the resilience of OpenPilot with an alert driver by conducting simulations both with and without attacks. Table 4.7 demonstrates that under normal system operation, without any attacks, no hazards or accidents occur. However, two *steer saturated* alerts were triggered due to the steering angle exceeding OpenPilot's predefined safety limits. Fig. 4.10 illustrates an example of the ALC system's performance. We observed that the

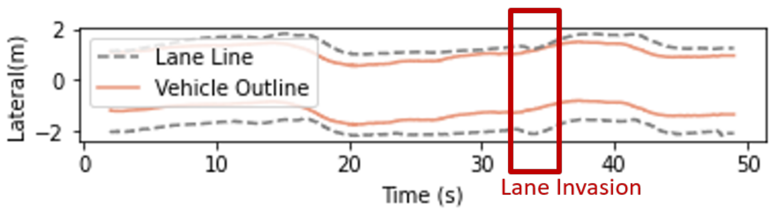


Figure 4.10: Trajectory of the Ego Vehicle during an attack-free simulation.

Table 4.7: Attack strategy comparisons with an alert driver.

Attack Strategy	**Sim.**	**Alerts**	**Hazards**	**Accident**	**Hazards& no Alerts**	**LaneInvasion (No. Event/s)**	**TTH(s) (Avg. ± Std.)**
No Attacks	240	2 (0.1%)	0	0	0	0.46	-
Random-ST+DUR	14,440	22.6%	39.8%	22.9%	21.4%	1.03	1.61±1.96
Random-ST	1,440	24.0%	53.5%	35.8%	32.9%	0.68	1.49±0.73
Random_DUR	1,440	14.6%	26.9%	23.1%	15.9%	0.46	1.92±1.17
Context-Aware	1,440	4 (0.3%)	**83.4%**	**44.5%**	**83.1%**	0.66	2.43±1.29

ALC system does not consistently keep the Ego vehicle centered in the lane, resulting in lane invasions at an average frequency of 0.46 times per second. This can lead to out-of-lane hazards or collisions with roadside objects. These findings indicate a lack of cooperation between the ALC and ACC systems, highlighting a defect in the control software that needs to be addressed.

Observation 1: Lane invasions can happen even without any attacks.

4.8.4 Evaluation of Attack Duration and Start Time

Table 4.7 reveals that the Context-Aware strategy surpasses the three random strategies, achieving the highest hazard coverage at 83.4% (1,201/1,440), with 99.7% (1,197/1,201) of hazards occurring without triggering any alerts. Notably, 53.4% (641/1,201) of hazards result in accidents, including collisions with the lead vehicle and roadside objects (see Fig. 4.9(c-d)). In these instances, the ADAS raises a *steer saturated* warning, while the more pertinent *forward collision* warning (FCW) is not activated because the brake output remains below OpenPilot's safety threshold. We also observe an increased number of lane invasions per second for almost all attacks due to the occurrence of out-of-lane hazards. Despite achieving the highest hazard coverage, the Context-Aware attacks maintain a low

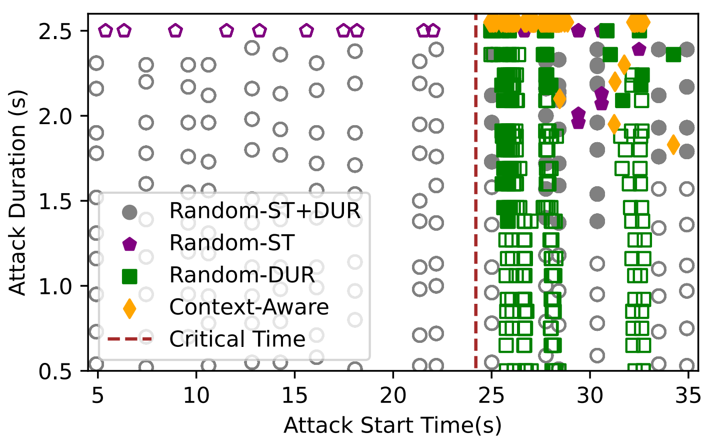


Figure 4.11: State space of “Attack start time" and “Duration" for *Acceleration* attacks (solid shapes correspond to hazardous results and empty ones to non-hazardous).

number of lane invasions and alerts due to strategic value corruption.

Observation 2: The Context-Aware attack strategy efficiently exploits safety-critical states of the ADAS. Notably, during these attacks, the *forward collision* warning (FCW) does not activate at all.

From Table 4.7, we also observe that the average Time-to-Hazard (TTH) of the Context-Aware attack is longer than that of the Random attacks. This is due to the higher hazard rate in the *Acceleration* attack, which has a longer TTH.

To further assess the significance of attack *duration* and *start* time, we analyzed the coverage of the fault parameter space by various attack strategies. Fig. 4.11 depicts a sample parameter space for durations ranging from 0.5 to 2.5 seconds and start times between 5 and 35 seconds for the *Acceleration* attack type. Each dot represents an attack simulation, with solid dots indicating those that resulted in hazards. The figure demonstrates that an attack does not induce any hazard if it is not activated within a specific time window (after the dashed line at approximately 24-25 seconds), regardless of its duration. Once the critical launch moment is identified, the attack must persist for a sufficient period (at least 1.5 seconds) to cause a hazard. Consequently, identifying both the optimal time to initiate an attack and the necessary duration is essential to increase the hazard success rate.

We also observe that the Context-Aware strategy (represented by orange diamonds) consistently results in hazards and falls within the critical time window. In contrast, the dots corresponding to the Random-ST and Random-DUR strategies show a substantial number of non-hazardous cases. This further highlights the efficiency of the proposed Context-Aware strategy.

Observation 3: The Context-Aware selection of start time and duration ensures that resources are not wasted on non-hazardous random injections.

4.8.5 Evaluation of Attack Value Selection: Control Outputs

In this series of experiments, we further assess the effectiveness of the Context-Aware strategy both with and without the use of strategic value selection.

Table 4.8 presents the results for different attack types, including the number of hazards mitigated by the driver. For effective hazard mitigation, the driver reaction times must be shorter than the Time-to-Hazard (TTH) (see average TTHs in Table 4.8). Our experiments show that without driver intervention, attacks without strategic value corruption can achieve very high hazard and accident success rates (nearly 100% for all attack types). However, when simulating human driver reactions, 83.3% of hazards are prevented for the *Acceleration* attack, resulting in a 50% reduction in collision events. Similar hazard reductions are observed for the *Deceleration* (58.8%) and *Deceleration-Steering* (70.8%) attacks.

Observation 4: Human alertness for timely intervention is important in preventing hazards and accidents.

However, driver reactions do not prevent *Steering* attacks, as evidenced by the zero hazards prevented for such attacks in Table 4.8. These attacks still achieve very high hazard and accident success rates, such as 100% for *Steering-Right* and *Acceleration-Steering*. This is because hazards occur in less than 1.63 seconds, significantly faster than the average human driver reaction time of 2.5 seconds. This indicates that attacks

Table 4.8: Context-aware attack with or without strategic value corruption and with an alert driver.

Attack Type	**W/o Strategic Value Corruption**							
	Alerts	**Hazards**	**Accident**	**TTH(s) (Avg. ± Std.)**	**Prevented Hazards**	**New Hazards**	**Prevented Accidents**	**Reduced Accidents**
Acceleration	4 (1.7%)	200 (83.3%)	120 (50.0%)	3.33±0.23	200 (83.3%)	160 (66.7%)	200 (83.3%)	120 (50%)
Deceleration	1 (0.4%)	99 (41.2%)	0 (0.0%)	2.62±0.04	141 (58.8%)	0	0	0
Steering-Left	122 (50.8%)	187 (77.9%)	175 (72.9%)	1.11±0.86	0	0	0	0
Steering-Right	2 (0.8%)	240 (100.0%)	240 (100.0%)	1.63±0.08	0	0	0	0
Acceleration-Steering	2 (0.8%)	240 (100.0%)	240 (100.0%)	1.51±0.15	0	0	0	-1 (0.4%)
Deceleration-Steering	3 (1.2%)	138 (57.5%)	17 (7.1%)	2.63±0.02	170 (70.8%)	68 (28.3%)	0	-17 (7.1%)
Total	142 (9.9%)	1104 (76.6%)	792 (55.0%)	2.04±1.10	511 (36.8%)	228 (16.4%)	200 (22.4%)	102 (11.4%)
	With Strategic Value Corruption							
Acceleration	1 (0.4%)	160 (66.7%)	160 (66.7%)	5.03±1.22	1	0	1	1
Deceleration	0 (0.0%)	231 (96.2%)	0 (0.0%)	2.77±0.10	0	0	0	0
Steering-Left	1 (0.4%)	90 (37.5%)	1 (0.4%)	1.33±0.17	0	0	0	0
Steering-Right	0 (0.0%)	240 (100.0%)	240 (100.0%)	1.39±0.10	0	0	0	0
Acceleration-Steering	2 (0.8%)	240 (100.0%)	240 (100.0%)	1.47±0.26	0	0	0	0
Deceleration-Steering	0 (0.0%)	240 (100.0%)	0 (0.0%)	2.77±0.06	0	0	0	0
Total	4 (0.3%)	1201 (83.4%)	641 (44.5%)	2.43±1.29	1	0	1	1

* The number of hazards/accidents prevented when a human driver simulator is added in the simulation.

targeting the steering angle are the most challenging for drivers to mitigate. It should be noted that *Steering-Left* attacks are less successful in causing hazards compared to *Steering-Right* attacks (77.9% vs. 100%) because the Ego vehicle starts in a lane closer to the right guardrail while traveling on a left-curved road.

Observation 5: Steering is the most effective attack type that cannot be easily halted by the human driver.

While driver intervention can reduce hazard and accident rates, it can also introduce new hazards. For instance, to avoid colliding with the vehicle in front, the Ego vehicle might stop in the middle of a lane, risking a rear-end collision or hitting curb objects. Table 4.8 reveals that up to 66.7% of new hazards occurred after preventing attacks on the gas output.

With the introduction of strategic value corruption, there's a 6.8% rise in hazard success rates (from 76.6% to 83.4%). Despite this increase, the total number of alerts generated by the ADAS drops to four, and the driver manages to prevent fewer than 0.1% of induced hazards, even when the average time to hazard (TTH) exceeds the average driver reaction time of 2.5 seconds (e.g., during *Acceleration*, *Deceleration*, and *Deceleration-Steering* attacks). This demonstrates the effectiveness of the Context-Aware strategy in avoiding detection by both the ADAS and human drivers.

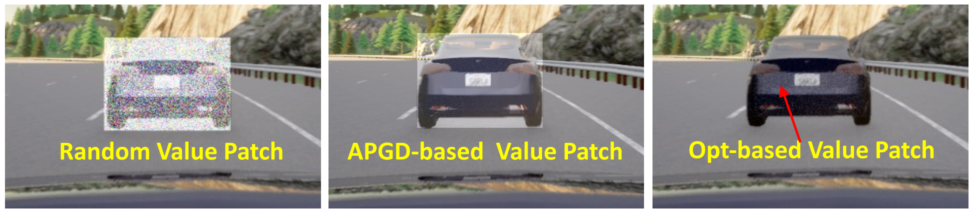
Observation 6: The strategic value corruption is effective in evading human driver detection and safety checks of ADAS.

4.8.6 Evaluation of Attack Value Selection: Perception Inputs

4.8.6.1 Attack Success Rate in Causing Hazards

To evaluate the effectiveness of the CA-Opt attack in causing safety hazards (**RQ1**), we perform experiments on the closed-loop simulation platform without activating safety interventions. This setup aligns with prior research [169, 170]. An attack is considered successful if a collision occurs (the Ego vehicle collides with the lead vehicle) or if the relative distance between the lead vehicle and the Ego vehicle is 0 meters or less. Unless otherwise specified, the success rate is reported based on 1,000 simulations.

Fig. 4.12 shows the success rates for each attack type. The CA-Random attack causes hazards in less than 1.2% of cases, with an overall success rate of 0.7%. This indicates that randomly generated adversarial patches have minimal impact on the DNN model's predictions and rarely lead to hazards. Even increasing the perturbation values in these patches does not significantly boost the success rate. This resilience is due to the DNN model in OpenPilot being trained primarily to detect the presence of front objects rather than classify them, making it less susceptible to adversarial attacks. Additionally, patches



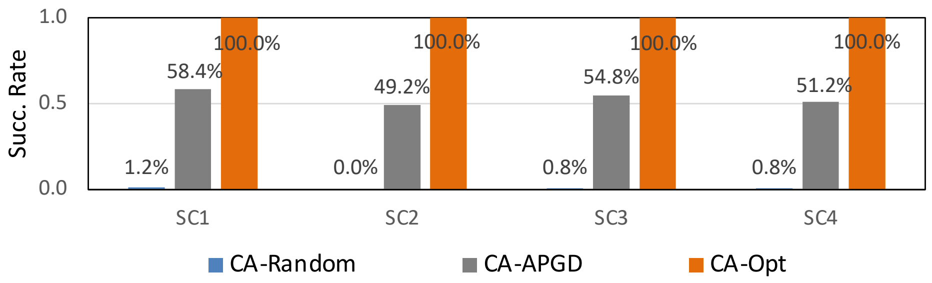


Figure 4.12: Top: Adversarial patch examples generated using our optimization method vs. random and APGD-based methods. Bottom: Success rate of CA-Opt and baseline attacks in absence of safety interventions.

with higher perturbation values are more noticeable to the driver than those created using the optimization-based method (Fig. 4.12-Top), which could alert the driver to prevent hazards.

Conversely, the proposed CA-Opt attack achieves a perfect success rate of 100% across all testing scenarios, outperforming CA-Random by a factor of 142.9. While CA-APGD uses a similar goal function as CA-Opt, it fails to cause hazards in 46.6% of simulations. This shortfall is primarily due to the limited number of iterations imposed to meet real-time constraints. In contrast, our CA-Opt attack employs a dynamic patch updating algorithm, which maintains the optimization process across perception cycles, thereby significantly enhancing the attack's effectiveness.

However, when the optimization-based perception attack is activated at random times and for random durations, it achieves an average success rate of just 3.5%. This is 28.6 times lower than the CA-Opt attack, as it wastes resources by injecting perturbations during non-critical system states. This underscores the importance of strategic timing of attacks and demonstrates that optimization-based methods alone are insufficient for causing hazards.

Observation 7: 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**).

4.8.6.2 Attack Stealthiness with Safety Interventions

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

Stealthiness in Perception Input. To evade detection by safety mechanisms and human drivers, the adversarial patch needs to remain as inconspicuous as possible. Generally, the smaller the pixel perturbations, the stealthier the attack. To evaluate this, we tested our attack method using three different λ values in Equation 4.4. We employed two sets of metrics to assess the patch's stealthiness: (i) the degree of pixel perturbation, measured using L_2 and L_∞ distances [216], and (ii) the similarity between the original camera image and the perturbed image, calculated using RMSE and the universal image quality index (UIQ) [2].

In Table 4.9, the results averaged over all test scenarios and simulations are presented. The CA-Opt attack maintains at least a 99.2% success rate under all three stealthiness levels, with perturbation degrees kept below 0.015 (L_∞) and 0.184 (L_2). The perturbed image with the adversarial patch has a UIQ similarity score of 0.993 compared to the original image, where 1 indicates identical images. For our evaluations, we selected a λ value of 10^{-3} due to its optimal balance of stealthiness and high attack effectiveness. Examples of the generated adversarial patches (with $\lambda = 10^{-3}$) are shown in Fig. 4.6 (see the zoomed-in area) and Fig. 4.12, and are nearly invisible to the human eye.

To further assess the stealthiness of our attack design, we conducted a user study with 30 participants. The results indicate that adversarial patches with $\lambda = 10^{-2}$ and $\lambda = 10^{-3}$ are almost imperceptible to human drivers. Additionally, the patches generated by CA-Opt attacks are less noticeable than those created by baseline perception attacks

Table 4.9: Attack success rate with different patch stealthiness levels λ and pixel perturbation degrees.

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.

(CA-Random and CA-APGD). Additional details can be found in Appendix B.

Evading Safety Interventions. To provide a more realistic evaluation of the effectiveness of different attack strategies, we reran our experiments with various safety interventions (introduced in Section 4.7). Before our evaluations, we calibrated the safety features to ensure the interventions were triggered correctly and without any false positives.

We test each attack method with different AEBS configurations: (i) FCW/AEB 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. 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 4.10 shows the experimental results of each attack method with different safety intervention configurations. We observe that, regardless of the interventions, the CA-

Table 4.10: 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 &	CA-Random	27.4%	0	100% (7/7)
AEBS Not Compromised	CA-APGD	100%	0	100% (534/534)
(Independent Camera)	CA-Opt	100%	**48.7%**	**51.3%** (513/1,000)
All &	CA-Random	24.3%	0	100% (7/7)
AEBS Compromised	CA-APGD	100%	0	100% (534/534)
(Shared Camera)	CA-Opt	14.6%	**89.6%**	**10.4%** (104/1,000)
All &	CA-Random	23.8%	0	100% (7/7)
AEBS Disabled	CA-APGD	100%	0	100% (534/534)
	CA-Opt	0	**100%**	**0**

Random and CA-APGD attacks fail to cause any hazards due to their low baseline success rates (see Fig. 4.12) 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 last row of Table 4.10), the CA-Opt attack successfully evades driver intervention in all simulations. This is attributed to the small, virtually invisible value of the adversarial patch, resulting in an average attack success rate of 100%. 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 4.7) to 10%, 5%, 2%, 1.5%, 1%, and 0.5%. As shown in Fig. 4.13, when perturbation thresholds are set to 0.5% and 1% (representing highly sensitive driver), the CA-Opt attack 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 a 100% success rate without triggering driver intervention. This finding underscores the robustness of the attack in evading driver detection across a range of driver sensitivities.

When the AEBS function is enabled and uses the same compromised camera inputs as ACC, CA-Opt attacks affect both the ACC and AEBS functionalities. So, the safety

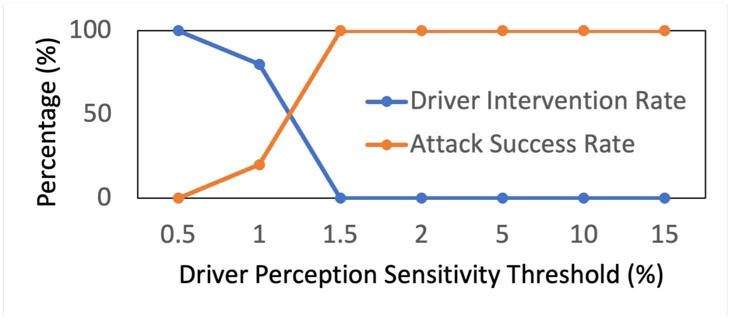


Figure 4.13: Attack success rate and driver intervention activation rate with different driver sensitivity thresholds.

interventions are triggered in only 14.6% of simulations, leading to a high success rate of 89.6%. 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 safety 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. 4.14) that delays AEBS activation and leaves insufficient time for hazard prevention.

Observation 8: 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 9: CA-Opt attack is more effective than baselines in keeping perturbations stealthy and causing hazards without being mitigated by safety interventions.

4.8.6.3 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 (❷ in Fig. 4.1) and ACC control output ❸. Although the attacker's goal is to maximize errors in the 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 the deviations resulting from the attack to those caused by stealthy attacks directly on DNN

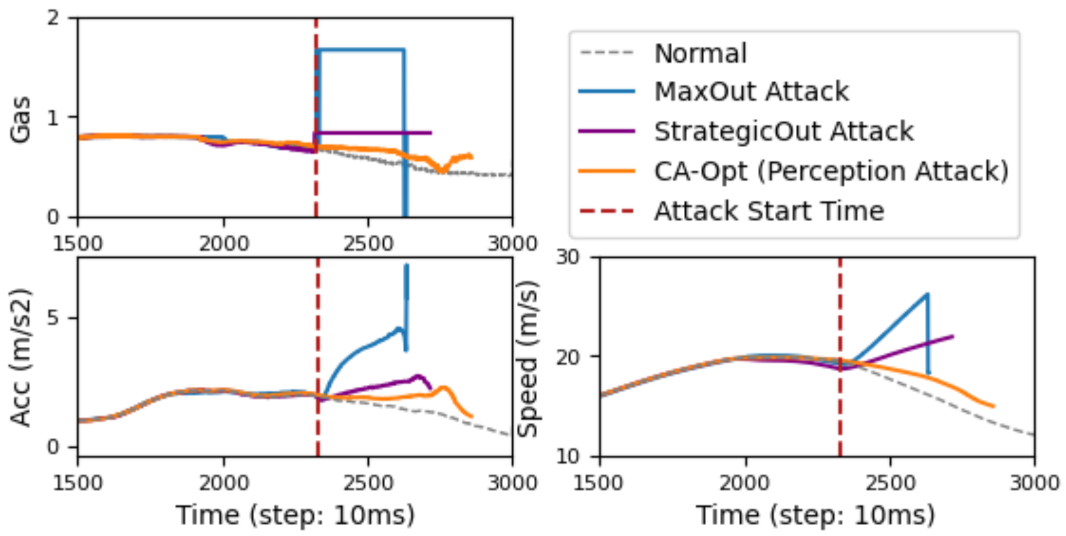


Figure 4.14: Context-Aware perception attacks vs. output attacks.

and control outputs. Note that such attacks are only possible under specific threat models (e.g., malware or wireless methods) in Table 4.3.

Control Output Attacks. We begin by examining deviations in the autonomous vehicle states and control outputs resulting from the attack compared to the proposed Context-Aware attack (see Section 4.8.2 and Table 4.6), which directly compromises the ACC control output to a strategic value (also referred to as **StrategicOut**). Additionally, we compare to another baseline, **MaxOut** which directly modifies ACC output control commands by setting them to the maximum allowed acceleration value. All these attacks employ the same context-aware method as CA-Opt for selecting the attack times and durations.

Fig. 4.14 illustrates an example scenario. The MaxOut attack leads to quicker 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, the perturbations injected by the CA-Opt attack into DNN perception inputs may not propagate to cause any changes in ACC output, or if they do cause changes, they will not be larger than the maximum possible acceleration caused by MaxOut attacks. These

Table 4.11: Performance of StrategicOut attack with all the safety features and different AEBS settings (AEBS with Shared Camera).

Attack Strategy	**Safety Interventions**	**Intervention Activation Rate**	**Succ. Rate**	**Hazard Prevention Rate**
StrategicOut	All & AEBS Activated	100%	20.3%	80.1% (797/995)
	All & AEBS Disabled	0.5%	99.5%	0.5%(5/1,000)
OptOut	All & AEBS Activated	100%	34.5%	65.5 (655/1,000)

perturbations lead to gradual deviations in system states over a longer period, achieving a high success rate (as shown in Fig. 4.12) 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 compared to the CA-Opt perception attack.

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 4.11 shows that without AEBS, StrategicOut achieves a high success rate of 99.5% 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%, primarily due to safety 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.

Similarly, we compare the performance of CA-Opt attack with a stealthy attack that

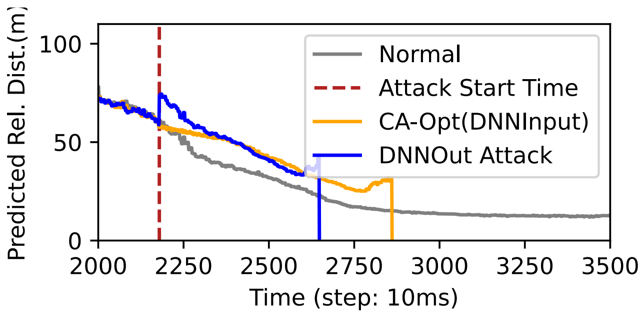


Figure 4.15: CA-Opt perception attack vs. DNN output attack.

directly compromises DNN output 2 (referred to as **DNNOut**) by formulating an optimization problem to maximize the relative distance (RD) prediction within one standard deviation while ensuring the satisfaction of safety constraints on acceleration and speed [111]. 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. 4.15) 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. 4.14.

Observation 10: 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.

4.8.6.4 Comparison to Fake Video Attacks

To further evaluate the necessity of a stealthy patch attack, we conduct another perception 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 4.8.2 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. 4.12) 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 11: 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. 4.16. Even if the fake videos are recorded using the same camera on the Ego vehicle driving in the same lane and

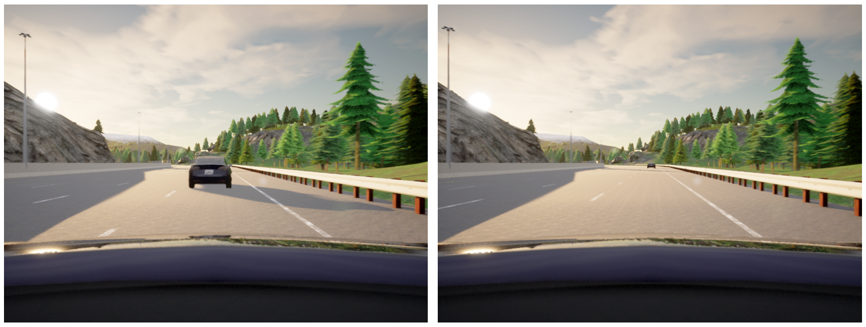


Figure 4.16: An example of two consecutive images before (Left) and after (Right) fake video attack.

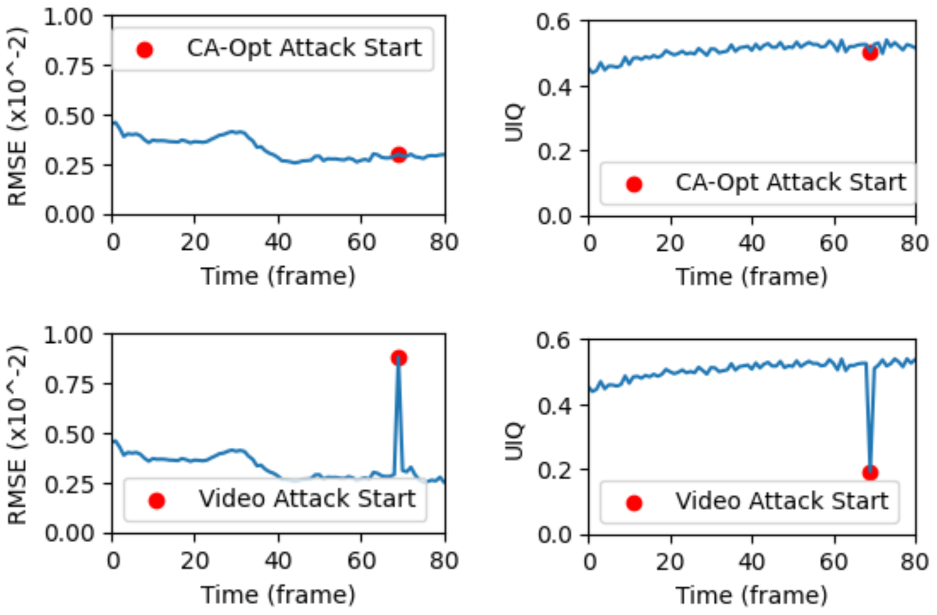


Figure 4.17: Similarity of two consecutive image frames with CA-Opt attack and fake video attack (starting at 69th frame) measured in RMSE and universal image quality index (UIQ) [2].

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 [2] between two consecutive image frames is shown in Fig. 4.17. 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 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 4.7).

4.9 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. We mainly assess the attack targeting the perception inputs as it covers the whole error propagation pipeline.

RQ4: Is our attack robust to real-world factors such as different camera positions and weather conditions?

RQ5: Can our attack transfer well from simulation to real-world implementation?

RQ6: How does our attack affect the real-time system operation?

RQ7: Can our attack evade detection or mitigation by the existing adversarial patch defense methods?

4.9.1 Robustness to Real-world Factors

To assess attack robustness, we vary front camera height based on standard passenger car profiles from manufacturers [217]. We perform our experiments with four heights between 1.1-1.7 meters and three initial distances (50m, 75m, 100m).

Fig. 4.18 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

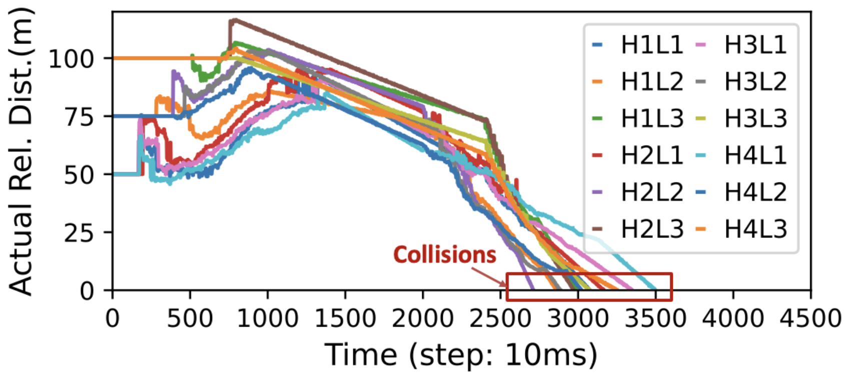


Figure 4.18: Actual relative distance trajectories under CA-Opt attack with different camera heights (H1:1.1m, H2:1.3m, H3:1.5m, H4:1.7m) and initial longitudinal distances to the lead vehicle (L1:50m, L2:75m, L3:100m). An actual relative distance of zero indicates collision.

deviations of 9.8-14.3m in the predicted lead vehicle position, while maintaining a success rate of 100% under such conditions.

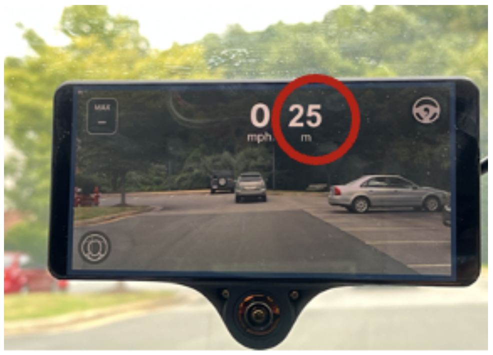
4.9.2 Performance on Actual Vehicles and Real-World Dataset

To evaluate the feasibility of the CA-Opt attack, we use a real-world dataset and an actual vehicle (Lexus NX 2020) equipped with a production L2 ADAS, Comma 3, running OpenPilot software v0.8.9. This evaluation examines the attack's impact on (i) the DNN perception module alone and (ii) the end-to-end ACC system.

4.9.2.1 Perception Module Evaluation

Actual Vehicle. We evaluated the perception module in two scenarios: (i) approaching a lead vehicle (LV) while parked in a parking lot and (ii) driving on an actual road. In each scenario, the ACC on the Ego vehicle was tested both with and without the adversarial patches injected into the camera frames.

In the first scenario, the Ego vehicle was parked at distances ranging from 10m to 50m (at intervals of 5m) from the LV. We modified the OpenPilot code to display the relative distance (RD) predictions on the device monitor, as shown in Fig. 4.19. In these tests, the CA-Opt attack caused an average deviation of 16.2m in distance predictions, which



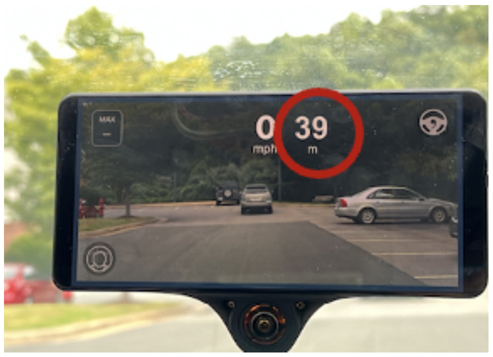


Figure 4.19: Evaluation on an actual car in a parking lot: relative distance predictions (Left) without and (Right) with adding an adversarial patch.

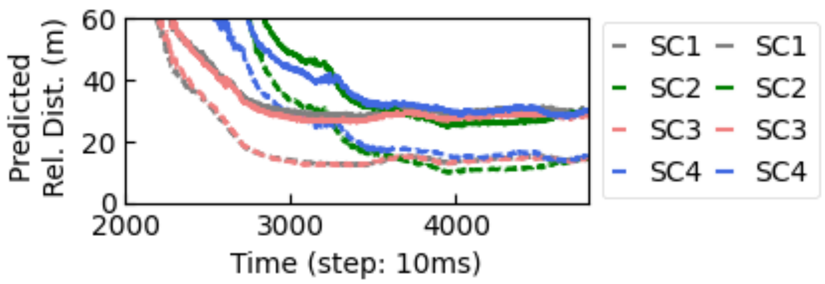


Figure 4.20: Relative distance predictions with (solid lines) or w/o (dashed lines) adversarial patch for different driving scenarios.

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 4.8.2. 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 (with an odd image index number) 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. 4.20, show that the attack significantly increased the relative distance predictions for 15.3m on average in all tested scenarios.

Testing on a Real-world Dataset. We also conducted experiments using a real-world video dataset, comma2k19 dataset [218], 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 selected 200 videos with a clear view of the lead vehicle and a relative distance of less than 100 meters. These videos were fed into the DNN model to record predictions for relative distance, which are considered as ground truth. We then introduced adversarial patches, generated by different attack methods, into the videos. The manipulated videos were again fed into the DNN model, and predictions for relative distance were compared with those from the videos without any attacks. Metrics such as the average and standard deviations in the predicted longitudinal distance were used for this comparison.

Table 4.12: Performance of CA-Opt attack vs. CA-Random 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

Table 4.12 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 4.6.2.1) in generating optimal perturbations with substantial impact. These experiments demonstrate the effectiveness of the CA-Opt attack in impacting the

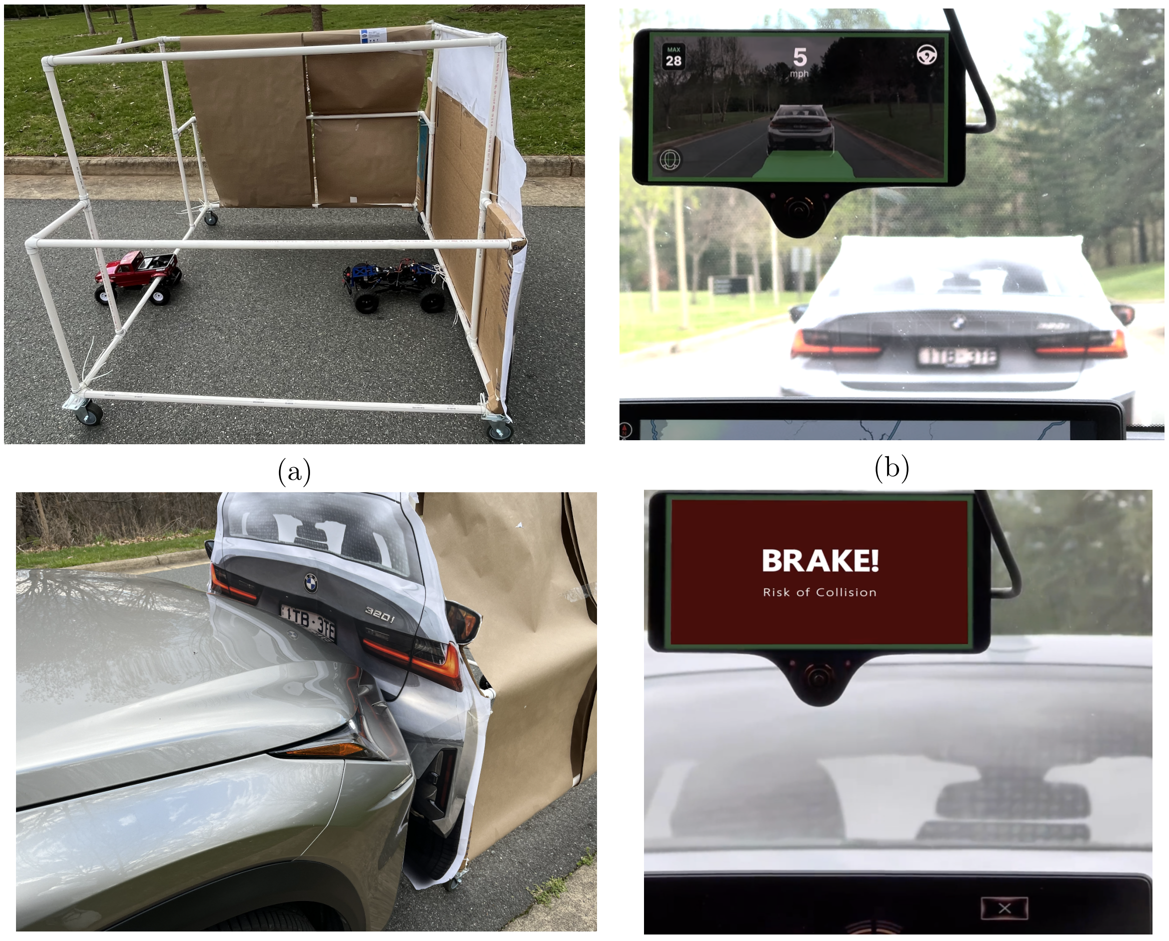


Figure 4.21: (a) Side view of lead car model; (b) AV follows the car model in a benign scenario; (c) AV under perception attack collides with the lead car model; (d) Driver's view upon collision.

DNN-based perception module in real-world driving scenarios.

4.9.2.2 End-to-End Evaluation

We also evaluate the impact of attacks on the end-to-end ACC on an actual vehicle [2]. 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 [219]. 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. 4.21-b). In this experiment, the AV approached the LV from a distance of 50 meters with a cruise speed set at 28

[2][Video: `https://sites.google.com/view/CAP-Attack`]

mph. Meanwhile, the LV was propelled by two remote-controlled ground robots (Fig. 4.21-a). We conducted this experiment with and without the 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. 4.21-b). However, in the presence of the attack, we observed that the AV continued to advance toward the lead car model and eventually collided with it (Fig. 4.21-c), despite the AEBS being activated (Fig. 4.21-d). 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 elapsed 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.

4.9.3 Runtime Overhead

To further evaluate the real-world applicability of our attack, we measured its runtime overhead on a Comma 3 device. The Ego vehicle, equipped with OpenPilot and our attack malware, was parked behind a lead vehicle in a parking lot with the ACC function activated and the cruise speed set to 0 mph. We recorded the time overhead for each component, as illustrated in Fig. 4.22, 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 [207] does not add significant overhead, leveraging gradients calculated during

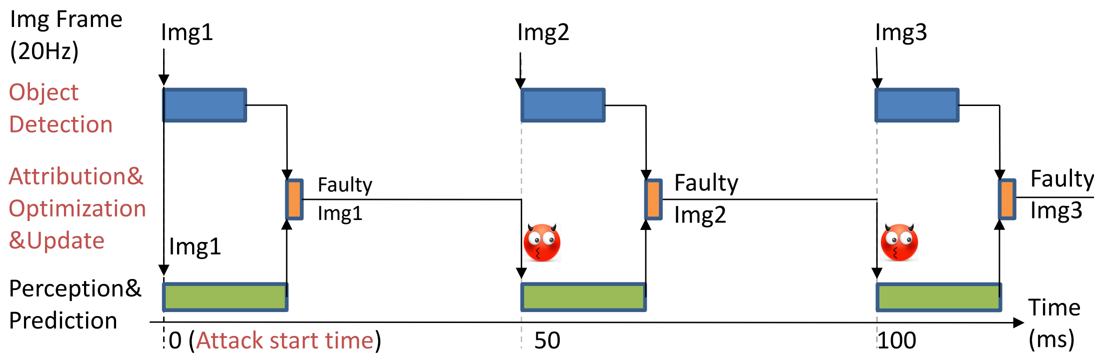


Figure 4.22: Runtime overhead of each step of the attack.

the patch optimization process. The total time overhead is 1.52 ms.

4.9.4 Evading Existing Defense Methods

While the proposed CA-Opt attack can create stealthy adversarial patches invisible to the human eye, it may be detected by some existing defense methods.

Adversarial Patch Detection. Methods such as gradient masking [220], lossy compression [221], or adversarial training [222] have been proposed for adversarial patch detection. However, these methods either need to be trained on specific attacks with high computation costs [223–225] or significantly sacrifice the DNN prediction accuracy [220, 226], which could negatively affect the safety of ACC systems.

We assess four widely used open-source defense methods that only rely on model input transformation without the need for re-training, including adding Gaussian noise [227], JPEG compression [228], reducing image color bit-depth [229], and using spatial median smoothing [229]. We evaluate the attack success rates in causing hazards under each defense method with various parameter settings 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. 4.23, JPEG compression and bit-depth reduction methods effectively reduce the attack success rate in causing hazards under specific parameter configurations.

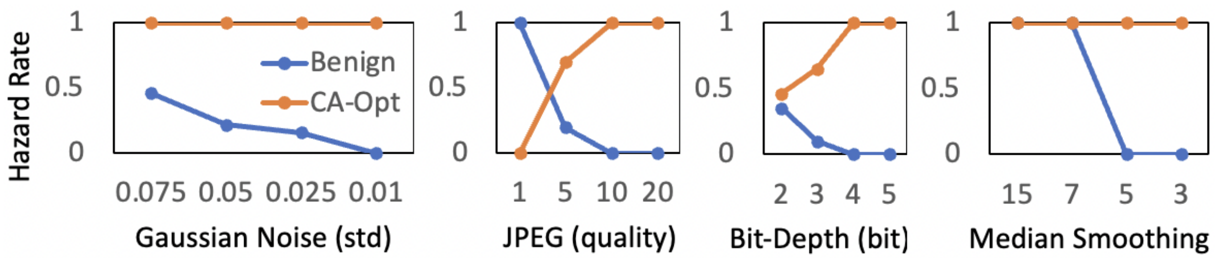


Figure 4.23: Results of each directly-applicable defense method.

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.

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 [170], and our tests found that radar-camera fusion did not prevent ACC misbehavior or collisions (see Appendix A.2.4). This may be because of the use of Kalman filters in sensor fusion, which assumes measurement noise is zero-mean Gaussian and are vulnerable to perturbations smaller than one standard deviation of this noise [157]. 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 typically carry more weight. The vulnerability of sensor fusion was also reported in previous works [230].

4.10 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 work, 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 [87]. Moreover, we evaluate the sim-to-real transfer possibility using an actual vehicle, a model lead car, and a publicly available dataset. However, limitations exist, such as the fixed model and thresholds used for the human driver simulator that may impact evaluation results.

Attack Method Generalization. We demonstrate the generalization of our proposed attack on a production ADAS, OpenPilot, through closed-loop simulation, real-world AV dataset, and actual vehicle experiments. However, the vulnerability of other Level-2 production ADAS, 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 OpenPilot ADAS. Specifically, our attack strategy, which leverages context awareness derived from high-level system hazard analysis, is not limited to specific systems and can be generalized to diverse ADAS. Furthermore, our optimization-based attack vector generation can be applied to other DNN-based ADAS, given the inherent vulnerability of DNNs to adversarial input perturbations [115,157,166,170] and the ADAS's inefficient resilience to control output attacks.

4.11 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 [115, 166,168,170,193,231–234]. 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 [169] 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 [235], GPS [236], radar [237], camera [238], lane detection [168, 170], multiple objects tracking [157, 167, 171], control software [111], and safety mechanisms [172]. To the best of our knowledge, this dissertation 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 4.4).

4.12 Conclusion

This work proposes a novel runtime strategic Context-Aware safety validation approach, which includes (i) a control-theoretic method to identify the most critical system contexts for launching attacks to maximize the likelihood of safety hazards, and (ii) a strategic attack value corruption method that targets control commands or uses an optimization-based image perturbation technique for efficiently generating and injecting adversarial patches into the DNN input. This approach aims to cause ADAS misbehavior and hazards as quickly as possible before detection or mitigation by ADAS safety mechanisms or human drivers. 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 various baselines. Our experimental results and observations indicate that steering is particularly vulnerable and that the current forward collision warning system is inadequate. This study also provides insights into the development of future ADAS that are robust against safety-critical attacks and highlights the importance of driver interventions and basic safety mechanisms in preventing such attacks.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This dissertation developed a holistic approach to context-aware assurance in CPS by combining knowledge and data driven techniques for runtime safety monitoring, hazard mitigation, and design time safety validation.

We first proposed a formal framework for the control-theoretic generation of safety context specification, which can be further refined using an STL learning method and synthesized into a safety engine for runtime safety assurance in CPS. The developed STL learning method learns the unknown scenario-specific parameters in the specified safety rules by minimizing a tight mean loss function, which captures the inter-scenario variability (e.g., different patient profiles or driving scenarios) and improves assurance performance.

We also developed combined knowledge and data driven methods for more accurate and timely hazard prediction and mitigation by integrating the generated safety context specification or general safety constraints into ML models using a custom loss function, which also enforces the satisfaction of safety requirements while maintaining the prediction and mitigation efficiency. In addition, we develop a path planning algorithm constrained by context-aware safety specifications and application-specific constraints, aiming to find the optimal mitigation path that can bring the system back to a state within a safe region as quickly and smoothly as possible. This optimal mitigation path will be fed to the developed ML model to derive a sequence of corrective control actions that can prevent potential hazards. A hazard time estimator is also developed to infer the deadline for launching the corrective actions and ensure safe recovery.

Finally, context-aware safety specifications are also used for the safety validation of CPS at design time. We propose a model-driven approach orthogonal to the traditional data-driven techniques which, instead of exploring the entirety of the fault parameter space, focuses on a systematic characterization of the effect of the timing of the faults (e.g., start time and duration of the faults) in conjunction with the dynamic state of the system to identify the most opportune system contexts for activation of faults. Upon determining the optimal times for initiating attacks, we proposed a strategic attack value generation method, targeting either the perception inputs or control outputs, with the goal of maximizing the chance of hazards and causing hazards as soon as possible, before being detected or mitigated by the human operators or the existing safety mechanisms.

We evaluated the proposed approaches by developing open-source closed-loop testbeds that integrate real-world control software and physical-world simulators together with typical safety mechanisms and a fault injection engine that simulates real-world adverse events reported in the literature. Experimental validation of the proposed safety assurance approaches for the applications of autonomous driving and smart health demonstrates their generalization to a broad range of CPS with improved accuracy, timeliness, and robustness. We hope this research can provide valuable designs, methodologies, and platforms for building safer attack-resilient CPS, open new research directions, and inspire more exciting work for future researchers.

5.2 Future Work

5.2.1 Research Vision

My research vision is that combining knowledge and data driven techniques will provide **automatic, adaptive, and trustworthy safety assurance** in safety-critical CPS. To reduce the complexity of generating safety context specifications, an automatic approach should be developed by combining domain knowledge and data-driven techniques. Moving towards adaptive safety monitoring requires detecting the changes in the system context and adapting the safety property and model accordingly. The integrated knowledge also helps build trustworthy safety assurance by offering a way to explain the ML model's outputs and enforcing satisfaction of safety properties in generating mitigation actions.

Automatic Safety Context Specifications. This work proposes a formal framework for the specification of the safety context that can be synthesized into a safety engine. However, it might still be time-consuming to generate such safety context specifications for a complex system. Thus, a more efficient way of generating these safety properties is desired, which includes the identification of critical state variables and the generation and refinement of safety rules. Previous work has studied the possibility of identifying critical state variables in a CPS using techniques such as program analysis [48, 239], statistical analysis, or ML-based methods [49]. In addition, many tools exist in the literature to generate safety rules, and this work also develops a method to learn the unknown parameters in the generated rules. Therefore, future work should focus on developing comprehensive tools and frameworks that enable the *automatic* identification of critical state variables, generation of safety rules, and learning of STL rules. This integration of techniques will streamline the process of safety context specification and contribute to the efficient and effective design of safety engines for complex systems.

Adaptive Safety Monitoring. Using ML and diverse and unpredictable human behaviors and activities pose new challenges to ensuring dependability, safety, and robustness. Further known CPS challenges are posed by system dynamics and uncertain environmental changes. Future direction on *adaptive* safety monitoring relies on formal specification and learning of *parametric* properties to capture critical contextual changes, adapt the parametric properties, the ML model, and the control actions according to different possible dynamics ranging from operational changes (humans with various skills and experiences) to physical and context changes (body physiology and system and environmental changes) [240, 241], and check the correctness of the system operation by verifying context-specific properties at run-time. Future work can explore the development of algorithms and techniques (such as out-of-distribution detection [242,243]) that can identify shifts in the underlying operational, physical, and environmental contexts. By detecting these changes, the safety monitoring system can adapt its monitoring parameters or activate specific safety measures to ensure the continued safety of the system. Another important aspect is the integration of online learning techniques into safety monitoring [244,245]. By focusing on detecting context changes and incorporating online learning techniques, adaptive safety monitoring can contribute to building more robust and responsive safety systems.

Trustworthy Safety Assurance. Deep Learning is a widely used and highly accurate technique in various CPS. However, the lack of transparency and interpretability in ML models has raised concerns, especially in safety-critical CPS, such as autonomous driving systems and medical CPS. In this work, I have taken the first steps towards improving the explainability of ML-based safety engines [47,53] by integrating safety context specifications into the ML model as a regularization item during the training process [131]. The integrated safety specifications serve as a way to explain or validate the ML models' outputs. Future work could explore techniques such as concept activation vectors [246] and model-agnostic explanations [247] to further improve the model's explainability. I also explored the enforcement of safety properties during the generation of recovery control actions [129]. By incorporating safety properties into the decision-making process, the generation of control actions can be guided to prioritize safety. Although these examples have limitations in their scope, they demonstrate the feasibility of combining knowledge and data-driven techniques to achieve trustworthy safety assurance.

My long-term research goal is that the proposed approach can be applied to a broader range of CPS and benefit the general public. This is possible given the generalization of the proposed approach in generating safety rules and integrating them into ML models. I have also implemented this approach in the applications of autonomous driving systems and artificial pancreas systems. I am interested in extending it to other CPS and fields, such as surgical robots, drones, ground robots, power grids, and industrial control systems. This research direction will not only improve the security of safety-critical CPS but also ensure the safety of system users such as autonomous vehicle drivers and diabetes patients.

5.2.2 Safety Validation and Assurance of ML-based CPS

ML technologies have also been increasingly used in safety-critical CPS for decision making because of their easier implementation, powerful capability in capturing the relationship between input and output or approximating dynamic models of the control systems, and high accuracy in predicting unseen data that shares similar features with the training data [248, 249]. Due to the complex environments within which ML-based CPS operate, ensuring their successful deployment is challenging. Great efforts have been made to ensure the success of assurance cases in traditional CPS, such as the published standards and guidelines for safety and effectiveness assessments (e.g., ISO 26262 [250], ISO/PAS 21448 [251], and FDA guideline for artificial pancreas systems [252]). However, none of the previous works [253–256] offer a safety assurance approach concerning ML techniques that instantiate the process activities and generate evidence for a concrete application in particular safety-critical CPS, such as the medical domain.

With the increasing use of ML technologies in the design of CPS and their safety engines, we also propose two approaches for ensuring the successful deployment of ML-based CPS, including a safety assurance case template for identifying the data and system requirements and an extended strategic safety validation method for evaluating the resilience of ML models against accidental or malicious perturbations on their inputs, inner architecture, or outputs.

Safety Assurance Case Template: We take APS as a representative case study of ML-based medical CPS and instantiate the proposed assurance case for a general framework of APS [257, 258], which is suitable for all types of APS. Our preliminary results on an analysis of the instantiated APS assurance case for an example ML-based APS controller proposed by Dutta et al. [135] shows that the proposed assurance case for the APS effectively characterizes the sufficiency of the data used during the ML controller's development in terms of relevance, completeness, accuracy, and balance [259].

Safety Validation of DNN Models in CPS: We present a method called Taylor-Guided Fault Injection (TGFI), a strategic fault injection technique that identifies and targets the most critical DNN weights for reliable inference. Using a modified Taylor criterion [260], we rank all DNN weights based on their relative importance to inference accuracy. By injecting faults into these crucial weights, we demonstrate that TGFI can efficiently uncover safety-critical vulnerabilities in autonomous vehicles [176].

Future Plan: After the successful deployment of ML within CPS, we plan to investigate the resilience of ML models against accidental or malicious perturbations on their inputs, inner architecture, and outputs. We will introduce small perturbations to the machine learning (ML) inputs or outputs using techniques similar to those proposed in our previous work [117]. For corrupting the inner ML architecture, we plan to strategically select a neuron within the model, preferably one that is more frequently utilized, and then flip a bit in its weight parameters. We will investigate how the faults at the neuron level affect the DNN predictions at the layer level as well as compromise the final control outputs at system level. Additionally, we intend to develop an efficient safety assurance method that focuses protection on the most critical neurons.

This section contains material from the previously published works [176, 259], coauthored with H. Alemzadeh, A. Schmedding, Y. Lu, L. Yang, P. Schowitz, E. Smirni, M. Bagheri, J. Lamp, and L. Feng, copyrighted by IEEE.

Appendix A

Design and Validation of Closed-loop CPS Testbeds

Most works in evaluating the safety validation approaches could be categorized into two directions: real-world experiment evaluation and simulation-based evaluation. Due to the high cost and risk of experiments in real applications, the simulation-based approach is preferred more by researchers as it allows considerable research on the safety assurance of CPS to be performed at an accelerated rate while avoiding unnecessary risk for system users. Thus, developing high-fidelity testbeds that can capture a variety of system dynamics as well as react to changes in the environment is very important. Another essential need is the ability to simulate unexpected events, such as accidental faults, human errors, and attacks that lead to adverse events. Such closed-loop testbeds can enable verification of control algorithms and safety features before the additional cost of real-world experiments and reduce the possibility of harm to actual system users and costly equipment.

The APS is a good example of a promising medical CPS that the U.S. Food and Drug Administration (FDA) approved through clinical trials. Much work has been done in the literature to develop realistic diabetes simulators [26,27,102], design advanced control algorithms to maintain glucose concentration at healthy levels [118,261], and conduct large-scale clinical trials [112,262,263]. However, to the best of our knowledge, none of these works considered closed-loop integration of simulators, controllers, and safety mechanisms

This appendix contains material from the previously published works [3,155], coauthored with H. Alemzadeh, M. Kouzel, A. Chen, H. Ren, M. McCarty, C. Nita-Rotaru, copyrighted by IEEE.

or the simulation of adverse events. Control software must be tested in a broad spectrum of environments and with a variety of patient profiles and physiological dynamics in order to be fully verified. Similar issues exist in the ADS literature. Although many testbeds have been proposed for the evaluation of ADS and autonomous vehicles [162, 177, 264], they either do not rely on real-world control software used on actual vehicles on the road or do not consider existing safety features in the cars or human driver interventions. Further, most previous works focus on level 3+ or fully autonomous vehicles [160], while almost all the commercial autonomous vehicles are still at level 2 [265, 266].

To fill this gap, this work presents the design and validation of open-source, closed-loop testbeds[1] for APS and ADS.

A.1 APS Testbed

The overall structure of the open-source closed-loop Artificial Pancreas System (APS) testbed is shown in Fig. A.1. The APS testbed includes two state-of-the-art glucose simulators (Glucosym simulator [79] and the UVA-Padova Type 1 Diabetes simulator [26]) and two control software (OpenAPS and Basal-Bolus), together with 40 virtual patients. The simulator can run with the integrated virtual patient library or by loading actual patient profiles. Similarly, the testbed also includes an extending interface to the controllers that can load external control algorithms to help improve or evaluate the controllers in commercial insulin pumps. Note that only one simulator and controller are selected to run the closed-loop simulation. We also design an adverse event simulator that can emulate common adverse events in APS, including hypoglycemic events, hyperglycemic events, diabetic ketoacidosis, or other device malfunctions (e.g., in CGM sensors, insulin pumps, or controllers), by injecting faults into the input/output of the control software at compile time.

The proposed closed-loop APS testbed and generated data traces are made publicly

[1]Available online at: `https://github.com/UVA-DSA/CPS-Runtime-Monitor`

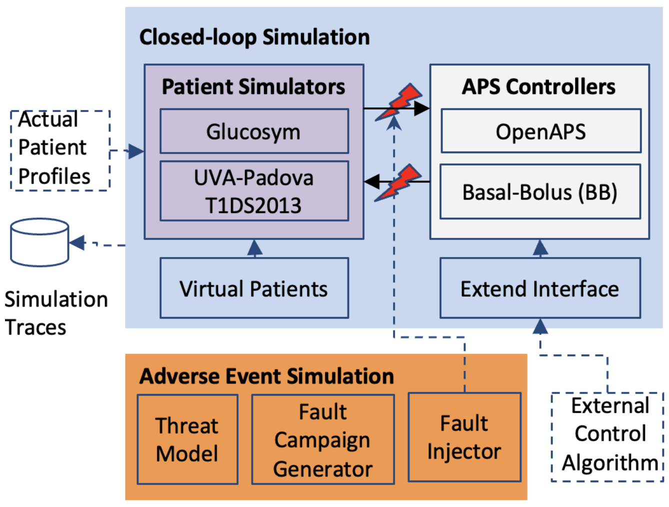


Figure A.1: Overall structure of the closed-loop APS testbed.

available to the research community[2]. The testbed is implemented with Python programming language at the application level, and can be installed on a Ubuntu operating system (16.04 LTS at least) automatically with an auto-script. This testbed offers a platform for other researchers to evaluate the performance of different control algorithms, validate the efficiency or safety of insulin delivery, develop the safety assurance or monitoring mechanisms for APS, and investigate the application of machine learning techniques in Type 1 diabetes treatment. The following subsections present a detailed description of the different components in the testbed.

A.1.1 Patient Glucose Simulators

Table A.1 shows an overview of the dynamic models used by each glucose simulator to emulate the effect of insulin dosage on the body, along with the required parameters for characterizing the patient profiles to run the simulators.

[2][Online Available: `https://github.com/UVA-DSA/APS_TestBed`]

Table A.1: Summary of patient glucose simulators.

Simulator	**Dynamic Model**	**Patient Profiles**
Glucosym	Medtronic Virtual Patient (MVP) Model: sub-cutaneous insulin delivery, the plasma insulin concentration, the insulin effect, the glucose kinetics, and the glucose appearance.	C_I, τ_1, τ_2, V_G, p_2, EGP, $GEZI$, S_I
UVA-Padova	Model of Kovatchev et al. [81]: plasma concentration, glucose fluxes, and insulin fluxes.	EGP, U_{ii}, U_{id}, k_1, k_2, G_{pb}

* C_I=Insulin clearance (dL/min).
* τ_1, τ_2=Time constant associated with insulin movement between the SC delivery site and plasma (min).
* V_G=Distribution volume in which glucose equilibrates (dL).
* p_2=Delay in insulin action upon increase in plasma insulin (1/min).
* EGP=Endogenous glucose production rate that would be estimated at zero insulin (mg/dL/min).
* GEZI=Effect of glucose per se to increase glucose uptake into cells and lower endogenous glucose production at zero insulin (1/min).
* S_I=Baseline sensitivity factor (dl/micro Unit).
* U_{ii}=Insulin-independent glucose utilization.
* U_{id}=Insulin-dependent glucose utilization.
* k_1, k_2=Rate parameters of glucose kinetics.
* G_{pb}=Initial amount of glucose in plasma.

A.1.1.1 Glucosym Patient Simulator

The Glucosym simulator is an open-source human body glucose simulator that was developed to help build and test automatic insulin delivery systems. This simulator contains patient models derived from data collected from 10 actual adult patients with Type I diabetes mellitus for 18 ± 13.5 years aged 42.5 ± 11.5 years, with their glucose dynamics predicted using a Medtronic virtual patient (MVP) model [102].

The MVP model includes five components that describe the sub-cutaneous insulin (I_{SC}) delivery, the plasma insulin concentration (I_P), the insulin effect (I_{EFF}) to lower blood glucose, the glucose kinetics, and the glucose appearance following a meal (R_A)

(see Eq. A.1-A.5). A three-compartment model [267] was used to identify the insulin activity after injection to the patient body (see Eq. A.1-A.3). With the value of glucose appearance given by the two-compartment model shown in Eq. A.5, the Bergman minimal model [268] and Sherwin model [269] described in Eq.A.4 were finally used to derive an estimation of the BG value at the next step. These five equations form the basis of the MVP dynamic model used in the Glucosym simulator for educating and training individuals with Type 1 diabetes [102]:

$$\frac{dI_{SC}(t)}{dt} = -\frac{1}{\tau_1} \cdot \left(I_{SC}(t) - \frac{ID(t)}{C_I} \right) \tag{A.1}$$

$$\frac{dI_P(t)}{dt} = -\frac{1}{\tau_2} \cdot (I_P(t) - I_{SC}(t)) \tag{A.2}$$

$$\frac{dI_{EFF}(t)}{dt} = -p_2 \cdot (I_{EFF}(t) - S_I \cdot I_P(t)) \tag{A.3}$$

$$\frac{dBG(t)}{dt} = -(GEZI + I_{EFF}(t)) \cdot BG(t) + EGP + R_A(t) \tag{A.4}$$

$$R_A(t) = \frac{C_H(t)}{V_G \cdot {\tau_m}^2} \cdot t \cdot e^{-\frac{1}{\tau_m}} \tag{A.5}$$

where, $GEZI, EGP, S_I, C_I, p2, \tau1, \tau2$ are patient-specific parameters, with their explanation presented in Table A.1. Other parameters, such as the input information of insulin doses and sampling frequency, are also needed for running the Glucosym simulator. The full list of input parameters used in this simulator is listed in Table A.2. An implementation of this simulator is publicly available at [79].

A.1.1.2 UVA-Padova Simulator

The other simulator we integrated into the APS testbed is the UVA-Padova Type 1 Diabetes Simulator, which FDA has approved for pre-clinical testing on animals. In this simulator, the model of glucose kinetics is described using the following equations [26]:

Table A.2: Input parameters of glucosym simulator.

Input	Description
Insulin Dose	Insulin dose in units given during the time-step. In the case of a basal (insulin delivery) adjustment, we need to calculate how much insulin will be given in the time-step defined by "dt" (i.e. how many insulin units will be given in 5 minutes by the set basal profile or temporary basal?).
dt	Change in time each step in minutes.
Index	Current index from the start of the simulation, starting at 0.
Time	Total simulation run-time in minutes.
Basal	The delivery of insulin.
Events	Events are set so that the simulator will consider them during the run. The events were sent on-the-go.

$$\frac{dG_p(t)}{d_t} = EGP - U_{ii} - k_1 G_p(t) + k_2 G_t(t)\ ,\ G_p(0) = G_{pb} \tag{A.6}$$

$$\frac{dG_t(t)}{d_t} = U_{id}(t) + k_1 G_p(t) - k_2 G_t(t)\ ,\ G_t(0) = G_{pb}\frac{k1}{k2} \tag{A.7}$$

where $G_p(t)$ represents the amount of glucose in plasma, and $G_p(t)$ describes the amount of glucose in the tissue. The blood glucose level that the CGM samples is given by Equation A.8:

$$G(t) = \frac{G_p(t)}{V_g} \tag{A.8}$$

The endogenous glucose production rate, EGP, is modeled as a function of glucose in plasma, $G_p(t)$, and delayed insulin action in the liver, $X^L(t)$, as shown in Equation A.9.

$$EGP = k_{p1} - k_{p2} \cdot G_p(t) - k_{p3} \cdot X^L(t) \tag{A.9}$$

$X^L(t)$ is based on insulin concentration in plasma. The insulin dose delivered to the patient by the pump, $ID(t)$, factors into this plasma insulin level via the insulin subsystem, which is split into $I_k(t)$ and $I_{sc}(t)$. $I_{sc}(t)$ represents the subcutaneous insulin

Table A.3: Input parameters of UVA-Padova simulator.

Input	**Description**
Initial BG	Starting value for patient's blood glucose
Sensor Settings	Type of CGM sensor and associated settings
Pump Settings	Type of insulin pump and associated settings
Meals	Sequence containing the time and size of each meal during the simulation
Profile	Unique parameters for the patient profile
Start Time	Beginning time for the simulation
Seed	Random number generator seed used for noise in sensor readings, etc.
Insulin Dose	Insulin dose to give to the patient for each step

level, and is impacted by insulin doses as follows:

$$\frac{dX^L(t)}{dt} = -k_i \cdot \left(X^L(t) - k_{ai}I_k(t) - k_{bi}I_{sc}(t)\right) \tag{A.10}$$

$$\frac{dI_{sc}(t)}{dt} = k_{sc}I_{sc}(t) + ID(t) \tag{A.11}$$

Other variables in the above equations are constant rate parameters that are part of the patient profile. This model was improved in 2013 by implementing the notion that insulin-dependent utilization increases non-linearly when glucose decreases below a certain threshold. Similar to the Glucosym simulator, the UVA-Padova simulator also uses the minimal glucose model to couple insulin action on glucose utilization and production. Other parameters required by the UVA-Padova simulator to run regularly are listed in Table A.3.

The two glucose simulators integrated with the APS testbed could also handle a single meal scenario for the virtual patient (VP) population, which is challenging for regulating BG in Type 1 diabetes because of unexpected human activities (e.g., meals or exercises) and patient variability (inter-patient and intra-patient).

Table A.4: Input parameters of OpenAPS.

Input	**Description**
Settings	Various settings specific to the pump
BG targets	High/low glucose targets set up in the pump
Insulin Sensitivity	The expected decrease in BG as a result of one unit of insulin
Basal profile	The basal rates that are set up in the pump
Preferences	User-defined preferences
Pump history	Last 5 hours data directly from the pump
Clock	Date and time that is set on the pump
Temp_basal	Current insulin delivery rate set up in pump
Glucose	Glucose level sensed by CGM

A.1.2 APS Controllers

We integrate two typical control algorithms into the APS testbed: a PID-based OpenAPS controller and a Basal-Bolus controller.

A.1.2.1 OpenAPS

OPenAPS is an advanced open-source control software used in the diabetes DIY community [118] that has comparable results with more rigorously developed and tested AP systems for glycaemic control [270] and is far safer than standard pump/CGM therapy with no reports of severe hypo- or hyperglycemic events [119].

The OpenAPS adjusts the insulin delivery of an infusion pump to automatically keep the BG level of the diabetic patient within a safe range. The internal architecture and necessary input-output connections of OpenAPS are shown in Fig. 2.4. The description of input parameters is listed in Table A.4. The shaded region indicates the OpenAPS controller, and the "File Storage" section reflects the behavior of the insulin pump. The functionality of OpenAPS can be divided into three processes. The *Get_profile* process accepts pump settings, target BG (BGT), insulin sensitivity, basal profile, and preferences as inputs and creates a profile required to calculate both IOB and recommended insulin delivery. The *Calculate_iob* process gets profile, clock, and pump history as input and

Algorithm 4: OpenAPS Algorithms

```
if $BG$ is rising, but $eventualBG < BG\_Target$ then
    cancel any temp basal;
else if $BG$ is falling, but $eventualBG > BG\_Target$ then
    cancel any temp basal;
else if $eventualBG > BG\_Target$ then
    cancel 30min temp basal;
    if recommended temp>existing basal then
        issue the new high temp basal;
    else if recommended temp<existing basal then
        issue the new high temp basal;
    else if 0 temp for >30m is required then
        extend zero temp by 30 min;
    end
end
```

calculates IOB. Finally, the *Determine_ basal* process accepts the profile, IOB, BG, and current insulin delivery (temp_basal) and calculates the suggested insulin delivery to the patient.

More specifically, OpenAPS collects the previously delivered insulin amount, combined with the duration of the activity, and it calculates the net IOB. Using the glucose sensor readings, OpenAPS then calculates the eventual BG using the following equation [271]:

$$eventualBG = CurrentBG - ISF * IOB + deviation \tag{A.12}$$

where $CurrentBG$ is the current BG, ISF is the Insulin Sensitivity Factor, and $EventualBG$ is the estimated BG by the end of current insulin delivery. A *deviation* term is also added, which is the difference in BG prediction based on purely insulin activity.

While the current BG is below a threshold value, OpenAPS continues to issue a temporary zero insulin delivery until the BG rises. Otherwise, OpenAPS determines whether the glucose values rise or fall more than expected. In that case, it performs the course of actions shown in Algorithm 4 [271].

Table A.5: Input parameters of Basal-Bolus controller.

Input	Description
CGM	Continuous glucose monitor sensor reading
CHO	Grams of carbohydrates consumed by patient (if meal occurred at current step)
BW	Patient's body weight
u_{2ss}	Steady state insulin rate per kilogram
CR	Insulin to carbs ratio
CF (ISF)	Insulin correlation (sensitivity) factor [272]

A.1.2.2 Basal-Bolus

Basal-Bolus regimens are widely used in insulin pumps [78, 112, 273]. Basal provides a constant supply of insulin to bring down high resting blood glucose levels. Bolus insulin, on the other hand, has a much more powerful but shorter-lived effect on blood sugar, making it an ideal supplement for people with diabetes to take after meals and in moments of extremely high blood sugar.

In the Basal-Bolus (BB) Controller, the constant supply of basal insulin is determined as shown in Equation A.13 [274]:

$$I_{basal} = \frac{u_{2ss} \cdot BW}{6000} \tag{A.13}$$

where u_{2ss} is the patient's steady-state insulin rate per kg and BW is body weight (kg), meaning basal insulin is in units of insulin per minute. Bolus insulin is determined by Equation A.14 when a meal has occurred (otherwise, no bolus is given) [274]:

$$I_{bolus} = \begin{cases} \frac{CHO}{CR} & \text{if } BG \leq 150 \\ \frac{CHO}{CR} + \frac{BG - BGT}{CF} & \text{if } BG > 150 \end{cases} \tag{A.14}$$

where CHO is the meal's size in grams of carbohydrates, BG is the CGM sensor reading, BGT is the target blood glucose of 120, CR is the insulin to carbs ratio, and CF is the correlation factor. The list of input parameters of the Basal-Bolus controller is also

Table A.6: Example recall event reports that involved device and software malfunctions.

Recall ID	Summary Recall Description	Cause	Affected Device
Z-1074-2013	The blood glucose meter will shut off and revert to set up mode at glucose values above 1023 instead of displaying EXTREME HIGH GLUCOSE.	Software Design	Glucose Monitor
Z-1034-2015	Calibration factors in the pump are overwritten during a programming step. The force sensor could send a lower signal value to the pump processor.	Software Design	Insulin Pump
Z-1734-2015	If the user does not act upon the E6 and E10 error messages appropriately, insulin delivery will be stopped and, if unnoticed, may lead to severe hyperglycemia.	Device Design	Insulin Pump
Z-1359-2012	An error was discovered in the blood glucose meter software so that the meter turns itself off when a user attempts to view results in the "Results Log" when the log has 256 or a multiple of 256 items to display.	Software Design	Glucose Monitor
Z-0929-2020	The mobile receiver can become stuck on the initialization screen when powering on. This will cause patients not to be able to receive glucose values or alerts	Software Design	Glucose Monitor
Z-1562-2020	The company identified potential interference from hydroxyurea. Patient use of the anti-neoplastic drug may falsely elevate glucose readings on the CGM.	Under Investigation	Gluocse Monitor
Z-2165-2020	After the device has been in use for about two months, data processing in the PDM can be slowed such that the Bolus Calculator fails to accurately subtract the correct amount of IOB before suggesting a bolus amount.	Device Design	Insulin Pump
Z-1772-2021	Under certain conditions, a software fault is detected when a large bolus delivery at a quick bolus speed completes. If the user is unaware of the amount of active insulin and delivers an additional bolus, there is a risk of insulin over delivery.	Software Design	Insulin Pump

summarized in Table A.5. This bolus is the units of insulin to be delivered, so it is divided by the length of a simulation step to become units of insulin per minute.

A.1.3 Closed Loop Simulation

Fig. 2.6 shows an example of the closed-loop simulation process by integrating the Glucosym simulator and OpenAPS control software. At each control loop, the estimated glucose value is updated and reported to the APS controller, based on which the controller calculates the recommended insulin dosage and sends it to the glucose simulator. The insulin amount is divided by 60 to convert the units from $Unit/hour$ to $Unit/minute$

to make OpenAPS and Glucosym work appropriately in a closed loop. The glucose value updates every five minutes (this is the value normally set by CGM [275]), and so does the control action.

In the UVA-Padova simulation, the CGM sensor is simulated by looking up the subcutaneous glucose state variable in the patient model, applying noise, and clipping it to be within the range of values an actual CGM sensor can return. Similarly, the simulated pump receives a basal and a bolus input from the controller, converts the values into the appropriate units ($pmol/min$), and clips the inputs to be within the real range of the insulin pump before sending the values to the patient model. These calls occur once per minute (5 times per environment step).

The Basal-Bolus controller uses additional patient-specific parameters to calculate insulin doses. For the basal insulin, it requires the patient's body weight and steady-state insulin rate. For the bolus dose, it uses the patient's insulin to carbs ratio (CR) and correlation factor (CF). Both CR and CF can be calculated from the Total Daily Dose (TDD) of insulin needed, which in turn is calculated from body weight, as shown in the following equations [26, 276]:

$$TDD = 0.55 \cdot BW \tag{A.15}$$

$$CR = 450/TDD \tag{A.16}$$

$$CF = 1700/TDD \tag{A.17}$$

A.1.4 Adverse Event Simulator

After a medical device, such as a CGM, insulin pump, or APS, is distributed in the market, the FDA monitors reports of adverse events and other problems with the device and, when necessary, alerts health professionals and the public to ensure proper use of the device and safety of patients [17, 277]. A recall is a voluntary action that a device manufacturer takes to correct or remove from the market any medical devices that violate the laws administrated by the FDA [278]. Recalls are initiated to protect public health and well-being from devices that are defective or that present health risks such as disease, injury, or death. In rare cases, if the company fails to recall a device that presents a health risk voluntarily, the FDA might issue a recall order to the manufacturer.

FDA regulations also require manufacturers to notify the FDA of the adverse events, including device malfunctions [279], serious injuries [91], and deaths [92] associated with medical devices. Not all reported adverse events lead to recalls. The device manufacturers and the FDA regularly monitor the adverse event reports to detect and correct problems in a timely manner.

Table A.6 shows example recall events from the FDA database where malfunctions of the commercially available APS devices or software were reported. The analysis and simulation of past recalls and typical adverse event scenarios can help with improving the design and test of the APS control algorithms and safety mechanisms and assessing their effectiveness in preventing similar adverse events [17, 280, 281]. However, it is too expensive and risky to simulate the adverse event scenarios with the actual patients and human operators in the loop due to the unacceptable consequences of adverse events and potential harm to patients.

To better evaluate the resilience of APS control algorithms against such safety issues, we design an adverse events simulator integrated with the closed-loop simulation. Specifically, we design a software-implemented fault injection (SWFI) engine (see Fig. A.1) that can automatically select a set of target locations within the APS software (e.g., variables representing the CGM sensor values and insulin dose commands) to inject faults (e.g., a zero value (*Truncate*), a previous value (*Hold*), or an arbitrary error value (*Add/Sub*)) and activate them under pre-defined trigger conditions and durations to mimic the typical adverse events listed in Table 2.2, including hyperglycemic (diabetic ketoacidosis) and hypoglycemic events, device malfunctions, and patient injuries. The adverse event simulator is an independent module and can be enabled or disabled manually.

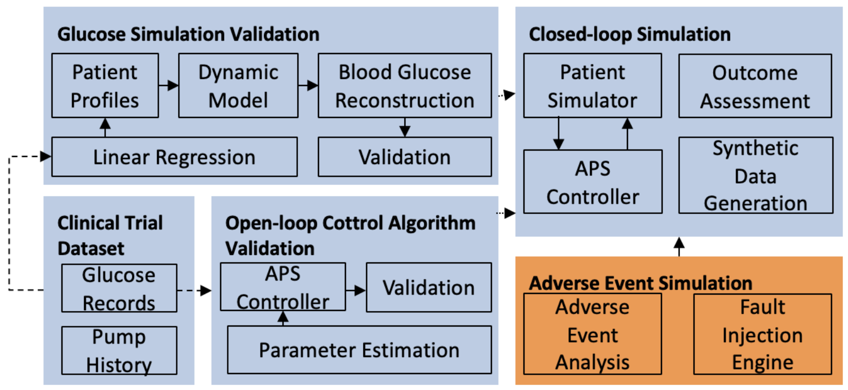


Figure A.2: Overall framework for validation of the APS testbed.

A.1.5 Validation of APS Testbed

We assess the validity of the proposed testbed by comparing the simulator outputs, controller outputs, and closed-loop outcomes with the data collected from a clinical trial (as shown in Fig. A.2). An optimization method is also proposed to reconstruct the blood glucose (BG) traces from a real-world clinical trial by estimating the patient profiles.

Our preliminary experimental results [3] show that the integrated glucose simulators can well reproduce the BG traces in the clinical trial, given the exact insulin dosages in the trial, and the integrated (open-loop) controllers keep a low mean squared error with the actual pump outputs in the clinical trial. The closed-loop simulations can keep blood glucose in a safe region 93.49% and 79.46% of the time on average, compared with 66.18% of the time for the clinical trial. Because the testbed aims to provide a platform to validate different control algorithms and safety features efficiently, we provide an easy-to-use fault injection implementation for both simulators and detail how the simulated faults compare to real-world fault scenarios in the device recalls and adverse events reported to the FDA.

A.2 ADS Testbed

We also developed a realistic closed-loop level-2 ADS testbed (introduced in Section 2.8.1.2 and Section 4.2.3) that integrates real-world control software used in over 250 car models driving on the road with the state-of-the-art CARLA urban driving simulator, a fault injection engine for simulation of faults/attacks, and a driver behavior simulator. To evaluate the safety of DNN-based ADAS systems under attack, we also enhanced the basic ADS testbed by developing a safety intervention simulator and mechanisms for priority-based dispatching of control commands to CARLA, as well as the fusion of camera and radar data. An overview of the enhanced simulation platform[3] is shown in Fig. 4.8, with the orange parts representing our new implementations.

A.2.1 AEBS (FCW and AEB) Simulator

We adopt and implement a time-to-collision (TTC) based phase-controlled AEBS [211] in our platform. The AEBS processes inputs derived from lead vehicle detection (LVD) outputs after sensor fusion, including relative distance (RD), relative speed (RS), and the current speed of the Ego vehicle (V_{Ego}) (see Fig. 4.1). The average driver reaction time (T_{react}) is standardized to 2.5 seconds, a commonly accepted value in the literature [111,170]. Various time thresholds are then computed: ttc (time to collision), t_{fcw} (forward collision warning time), t_{pb1} (first phase partial brake time), t_{pb2} (second phase partial brake time), and t_{fb} (full brake time) as follows:

$$ttc = RD/RS \tag{A.18}$$

$$t_{fcw} = T_{react} + V_{Ego}/4.5 \tag{A.19}$$

$$t_{pb1} = V_{Ego}/2.8; t_{pb2} = V_{Ego}/5.8; t_{fb} = V_{Ego}/9.8 \tag{A.20}$$

As shown int Fig. A.3, 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 value blocks other controls from the ADAS.

[3][Online Available: `https://github.com/UVA-DSA/openpilot-CARLA`]

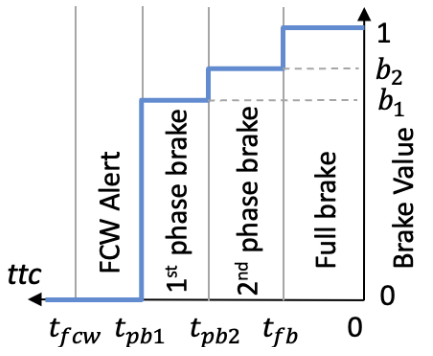


Figure A.3: AEBS.

Table A.7: 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 [209], we employ two categories and five driving scenarios (see Table A.7) to assess our AEBS functionality. 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 approximately 1.68 seconds for AEB to stop the Ego vehicle completely.

A.2.2 Driver Reaction Simulator

To evaluate driver interventions, we develop a driver reaction simulator. The simulated driver receives notifications when any ADAS safety alerts are triggered (e.g., FCW) or when the driver notices anomalies in the vehicle's status or camera user interface (UI). These anomalies include hard braking ($|Brake| > |limit_{brake}|$), unexpected increases in acceleration ($Accel > limit_{accel}$) or steering ($Steering > limit_{steer}$), the vehicle speed exceeding the cruising speed by more than 10% ($Speed > 1.1v_{cruise}$), or the mean perturbation value in the UI exceeding a noticeable threshold, set by default to 15% for an alert driver ($Patch.mean() > 0.15$). Assuming a highly alert driver capable of detecting

Table A.8: Driver simulator: activation conditions and reactions.

Activation Condition	**Driver Reaction**	**Reaction Time**
Alerts (e.g., FCW) Unexpected Acceleration Unexpected Steering Unsafe Cruise Speed Obvious Camera Perturbation	Emergency Brake (Eq. A.21) 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

anomalies within a single control cycle (10ms), the driver issues a predefined emergency response (see Table A.8) that takes effect 2.5 seconds later, which is the average driver reaction time.

In response to sudden unintended acceleration, human drivers typically apply a hard brake within 1.5 seconds. We model this behavior using an exponential function that approximates the general brake curve, as follows [282]:

$$brake = e^{10t-12}/(1 + e^{10t-12}) \tag{A.21}$$

We apply the same reaction model to sudden steering changes. The attack engine ceases its actions immediately upon driver intervention.

A.2.3 Priority-based Control Command Dispatcher.

With multiple safety mechanisms in place, there might be conflicts among the control commands issued by the OpenPilot ADAS 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., ADAS, AEB, simulated driver) based on their priorities, with high-priority commands overwriting low-priority ones (see Fig. 4.8 and Fig. A.4). The simulated driver's actions have a higher priority than regular ADAS outputs, and control actions from the AEB have the highest

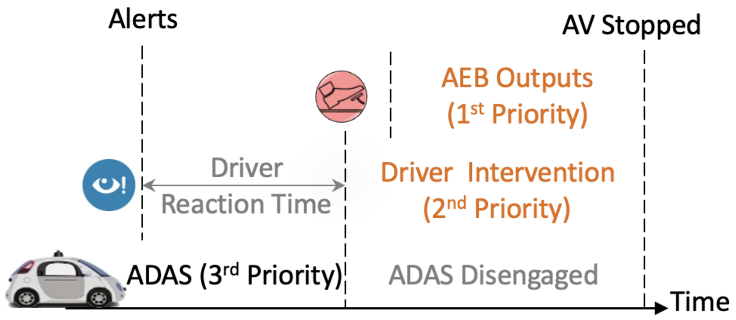


Figure A.4: Control command dispatcher.

priority. The driver's actions will be executed 2.5s (average driver reaction time) after safety alerts (e.g., FCW) or noticing other ADAS malfunctions (see Table A.8). ADAS commands will be blocked or disengaged when AEB or driver interventions are triggered.

A.2.4 Sensor Fusion

We implement a radar sensor in the CARLA simulator and feed the data to the OpenPilot radar interface [283], to be used as an independent input by the fusion module. Specifically, we use the DBSCAN algorithm [284] 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. A.5 (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. A.5 (Bottom) and discussed in Section 4.9.4, even though it fails to prevent collisions in the end.

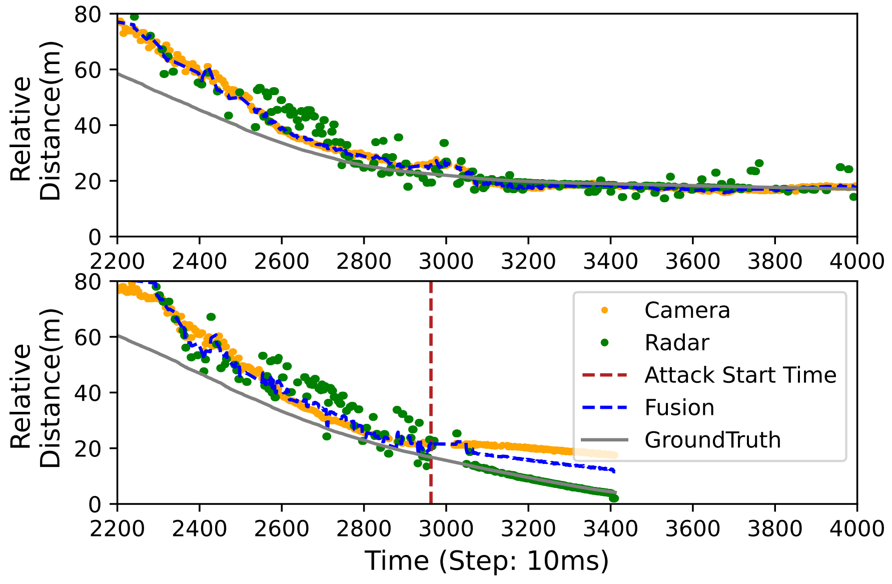


Figure A.5: An example fusion of relative distance predictions based on camera and radar data compared with the ground truth under normal operation (Top) or under attack (Bottom).

Appendix B

Stealthiness User Study

We conduct a user study [285] 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 4.2.3). 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 [286]. All participants had driving experience and 40% of them had autonomous 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. B.1 show that 99% of the participants express a preference for 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

This appendix contains material from the previously published works [155], coauthored with H. Alemzadeh, M. Kouzel, A. Chen, H. Ren, M. McCarty, C. Nita-Rotaru, copyrighted by IEEE.

for a larger monitor size or a more prominent 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.

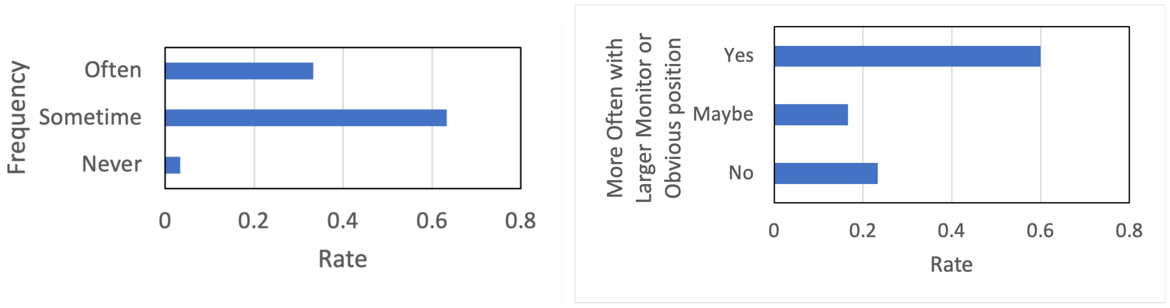


Figure B.1: Results of participants' preference of looking at the ADAS monitor during driving and whether they would look more often at the monitor with a larger size or in a noticeable position.

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 4.8.6.2). 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.

Results of the user study are illustrated in Fig. B.2. 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 that 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.

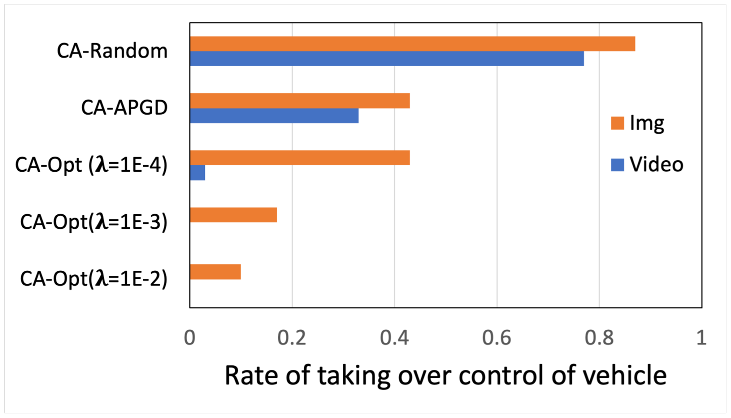


Figure B.2: Results of stealthiness of each attack method.

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 [169]. 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.

Bibliography

[1] E. Scharwächter and E. Müller, "Statistical evaluation of anomaly detectors for sequences," 2020. [Online]. Available: https://arxiv.org/abs/2008.05788

[2] Z. Wang and A. C. Bovik, "A universal image quality index," *IEEE signal processing letters*, vol. 9, no. 3, pp. 81–84, 2002.

[3] X. Zhou, M. Kouzel, H. Ren, and H. Alemzadeh, "Design and validation of an open-source closed-loop testbed for artificial pancreas systems," in *2022 IEEE/ACM Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*. IEEE, 2022, pp. 1–12.

[4] U.S. Food and Drug Administration, "Class 2 device recall medtronic minimed paradigm veo insulin pump," https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfres/res.cfm?id=175809, March 26, 2020.

[5] K. Kim, J. S. Kim, S. Jeong *et al.*, "Cybersecurity for autonomous vehicles: Review of attacks and defense," *Computers & Security*, vol. 103, p. 102150, 2021.

[6] Keen Security Lab of Tencent, "New car hacking research: 2017, remote attack tesla motors again." 2017. [Online]. Available: https://keenlab.tencent.com/en/2017/07/27/New-Car-Hacking-Research-2017-Remote-Attack-Tesla-Motors-Again/

[7] Z. Chris, "A google self-driving car caused a crash for the first time," *The Verge*, 2016.

[8] B. Broadhead, B. Rearden, C. Hopper, J. Wagschal, and C. Parks, "Sensitivity- and uncertainty-based criticality safety validation techniques," *Nuclear science and engineering*, vol. 146, no. 3, pp. 340–366, 2004.

[9] G. Philip, M. Dsouza, and V. Abidha, "Model based safety analysis: Automatic generation of safety validation test cases," in *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*. IEEE, 2017, pp. 1–10.

[10] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming dr. frankenstein: Contract-based design for cyber-physical systems," *European Journal of Control*, vol. 18, no. 3, pp. 217–238, 2012.

[11] H. Ziade, R. A. Ayoubi, R. Velazco *et al.*, "A survey on fault injection techniques," *Int. Arab J. Inf. Technol.*, vol. 1, no. 2, pp. 171–186, 2004.

[12] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, 1997.

[13] A. Rao, N. Carreon, R. Lysecky, and J. Rozenblit, "Probabilistic threat detection for risk management in cyber-physical medical systems," *IEEE Software*, vol. 35, no. 1, pp. 38–43, 2017.

[14] N. Leveson and J. Thomas, “An stpa primer,” *Cambridge, MA*, 2013.

[15] J. Wan, A. Canedo, and M. A. Al Faruque, “Functional model-based design methodology for automotive cyber-physical systems,” *IEEE Systems Journal*, vol. 11, no. 4, pp. 2028–2039, 2017.

[16] F. Cairoli, G. Fenu, F. A. Pellegrino, and E. Salvato, “Model predictive control of glucose concentration based on signal temporal logic specifications,” in *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*, 2019, pp. 714–719.

[17] H. Alemzadeh, R. K. Iyer, Z. Kalbarczyk, and J. Raman, “Analysis of safety-critical computer failures in medical devices,” *IEEE Security & Privacy*, vol. 11, no. 4, pp. 14–26, 2013.

[18] C. C. Oliveira and J. M. da Silva, “A fuzzy logic approach for highly dependable medical wearable systems,” in *IMSTW*, 2015, pp. 1–5.

[19] M. Ahmed, A. N. Mahmood, and J. Hu, “A survey of network anomaly detection techniques,” *Journal of Network and Computer Applications*, vol. 60, pp. 19–31, 2016.

[20] H. Choi, W.-C. Lee, Y. Aafer *et al.*, “Detecting attacks against robotic vehicles: A control invariant approach,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 801–816.

[21] M. R. Aliabadi, A. A. Kamath, G.-S. Julian *et al.*, “ARTINALI: dynamic invariant detection for cyber-physical system security,” in *ESEC/FSE 2017*. ACM, 2017, pp. 349–361.

[22] A. A. Cárdenas, S. Amin, Z.-S. Lin *et al.*, “Attacks against process control systems: Risk assessment, detection, and response,” in *ASIACCS*, 2011, p. 12.

[23] L. Zhang, X. Chen, F. Kong, and A. A. Cardenas, “Real-time attack-recovery for cyber-physical systems using linear approximations,” in *2020 IEEE Real-Time Systems Symposium (RTSS)*, 2020, pp. 205–217.

[24] L. Zhang, K. Sridhar, M. Liu, P. Lu, X. Chen, F. Kong, O. Sokolsky, and I. Lee, “Real-time data-predictive attack-recovery for complex cyber-physical systems,” in *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2023, pp. 209–222.

[25] H. Lin, H. Alemzadeh, C. Daniel, K. Zbigniew, and K. I. Ravishankar, “Safety-critical cyber-physical attacks: Analysis, detection, and mitigation,” *Symposium and Bootcamp on the Science of Security (HOTSOS)*, 2016.

[26] C. D. Man, F. Micheletto, D. Lv, M. Breton, B. Kovatchev, and C. Cobelli, “The uva/padova type 1 diabetes simulator: new features,” *Journal of diabetes science and technology*, vol. 8, no. 1, pp. 26–34, 2014.

[27] R. Visentin, E. Campos-Náñez, M. Schiavon, D. Lv, M. Vettoretti, M. Breton, B. P. Kovatchev, C. Dalla Man, and C. Cobelli, "The uva/padova type 1 diabetes simulator goes from single meal to single day," *Journal of diabetes science and technology*, vol. 12, no. 2, pp. 273–281, 2018.

[28] L. Masson, J. Guiochet, H. Waeselynck, A. Desfosses, and M. Laval, "Synthesis of safety rules for active monitoring: application to an airport light measurement robot," in *2017 First IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2017, pp. 263–270.

[29] W. Young, J. Corbett, and M. S. Gerber et al., "Damon: A data authenticity monitoring system for diabetes management," in *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 2018.

[30] C. Hernandez and J. Abella, "Timely error detection for effective recovery in light-lockstep automotive systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 11, pp. 1718–1729, 2015.

[31] H. Choi, S. Kate, Y. Aafer, X. Zhang, and D. Xu, "Software-based realtime recovery from sensor attacks on robotic vehicles," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*. San Sebastian: USENIX Association, Oct. 2020, pp. 349–364.

[32] M. Yasar and H. Alemzadeh, "Real-time context-aware detection of unsafe events in robot-assisted surgery," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020, pp. 385–397.

[33] P. Dash, G. Li, Z. Chen, M. Karimibiuki, and K. Pattabiraman, "Pid-piper: Recovering robotic vehicles from physical attacks," in *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 26–38.

[34] A. Singla, E. Bertino, and D. Verma, "Overcoming the lack of labeled data: Training intrusion detection models using transfer learning," in *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2019, pp. 69–74.

[35] Y. Zhou and K. Murat, "On transparency of machine learning models: A position paper," in *AI for Social Good Workshop*, 2020.

[36] T. Ouyang, V. S. Marco, Y. Isobe, H. Asoh, Y. Oiwa, and Y. Seo, "Corner case data description and detection," in *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. IEEE, 2021, pp. 19–26.

[37] K. R. Varshney and H. Alemzadeh, "On the safety of machine learning: Cyber-physical systems, decision sciences, and data products," *Big data*, vol. 5, no. 3, pp. 246–255, 2017.

[38] H. Alemzadeh, D. Chen, X. Li *et al.*, "Targeted attacks on teleoperated surgical robots: Dynamic model-based detection and mitigation," in *2016 46th Annual IEEE/IFIP International Conference on DSN)*. IEEE, 2016, pp. 395–406.

[39] A. Roederer, J. Dimartino, J. Gutsche, M. Mullen-Fortino, S. Shah, C. W. Hanson, and I. Lee, "Clinician-in-the-loop annotation of icu bedside alarm data," in *2016 IEEE first international conference on connected health: applications, systems and engineering technologies (CHASE)*. IEEE, 2016, pp. 229–237.

[40] N. Leveson, *Engineering a safer world: Systems thinking applied to safety.* MIT press, 2011.

[41] H. Lin, H. Alemzadeh, Z. Kalbarczyk, and R. Iyer, "Challenges and opportunities in the detection of safety-critical cyberphysical attacks," *Computer*, vol. 53, no. 3, pp. 26–37, 2020.

[42] G. Sannino and G. D. Pietro, "A mobile system for real-time context-aware monitoring of patients' health and fainting," *International Journal of Data Mining and Bioinformatics*, vol. 10, no. 4, p. 407, 2014.

[43] R. Ivanov, J. Weimer, and I. Lee, "Context-aware detection in medical cyber-physical systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, 2018, pp. 232–241.

[44] P. Asare, J. Lach, and J. A. Stankovic, "Fstpa-i: A formal approach to hazard identification via system theoretic process analysis," in *ICCPS*, April 2013, pp. 150–159.

[45] J. Thomas, "Extending and automating a systems-theoretic hazard analysis for requirements generation and analysis," *Technical Report SAND2012-4080*, 2012.

[46] R. Quinonez, J. Giraldo, L. Salazar, E. Bauman, A. Cardenas, and Z. Lin, "SAVIOR: Securing autonomous vehicles with robust physical invariants," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 895–912.

[47] X. Zhou, B. Ahmed, J. H. Aylor, P. Asare, and H. Alemzadeh, "Data-driven design of context-aware monitors for hazard prediction in artificial pancreas systems," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 484–496.

[48] A. Ding, P. Murthy, L. Garcia, P. Sun, M. Chan, and S. 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*, 2021, pp. 428–441.

[49] J. Saurabh, S. B. Subho, T. Timothy, K. S. H. Siva, and B. S. Michael, "Ml-based fault injection for autonomous vehicles: A case for bayesian fault injection," in *Proceedings of the 49th IEEE/IFIP International Conference on Dependable Systems and Networks.* IEEE, 2019, pp. 112–124.

[50] S. Jha, T. Tsai, S. Hari, M. Sullivan, Z. Kalbarczyk, S. W. Keckler, and R. K. Iyer, “Kayotee: A fault injection-based system to assess the safety and reliability of autonomous vehicles to faults and errors,” *arXiv preprint arXiv:1907.01024*, 2019.

[51] R. H. Bishop and R. C. Dorf, *Modern control systems*, 2011.

[52] E. F. Camacho and C. B. Alba, *Model predictive control.* Springer science & business media, 2013.

[53] X. Zhou, B. Ahmed, J. H. Aylor, P. Asare, and H. Alemzadeh, “Hybrid knowledge and data driven synthesis of runtime monitors for cyber-physical systems,” *IEEE Transactions on Dependable and Secure Computing*, 2023.

[54] A. Mehmed, W. Steiner, and A. Causevic, “Formal verification of an approach for systematic false positive mitigation in safe automated driving system,” *Mälardalen Real-Time Research Centre, Mälardalen University*, April 2020.

[55] Y. Luo, Y. Xiao, L. Cheng, G. Peng, and D. D. Yao, “Deep Learning-based Anomaly Detection in Cyber-physical Systems: Progress and Opportunities,” *ACM Computing Surveys*, vol. 54, no. 5, pp. 106:1–106:36, May 2021.

[56] U.S. Food and Drug Administration, *Policy for Device Software Functions and Mobile Medical Applications*, https://www.fda.gov/media/80958/download, 2019.

[57] H. Krasner, “The cost of poor quality software in the us: A 2018 report,” *Consortium for Information & Software Quality*, 2018.

[58] Virginia DMV, “Safe driving,” https://www.dmv.virginia.gov/webdoc/pdf/dmv39d.pdf.

[59] A. Desai, S. Ghosh, S. A. Seshia, N. Shankar, and A. Tiwari, “Soter: A runtime assurance framework for programming safe robotics systems,” in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 138–150.

[60] “Safety envelope requirements approach,” https://www.youtube.com/watch?v=_wxlgbl4s-g.

[61] P. Asare, “A Framework for Reasoning about Patient Safety of Emerging Computer-Based Medical Technologies,” Ph.D. dissertation, University of Virginia, May 2015.

[62] E. Bartocci, “Monitoring, learning and control of cyber-physical systems with stl (tutorial),” in *Runtime Verification*, C. Colombo and M. Leucker, Eds. Cham: Springer International Publishing, 2018, pp. 35–42.

[63] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, and S. Sankaranarayanan, “Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications,” in *Lectures on Runtime Verification.* Springer, 2018, pp. 135–175.

[64] B. Zhang and W. Zuo, “Learning from positive and unlabeled examples: A survey,” in *2008 International Symposiums on Information Processing*, May 2008, pp. 650–654.

[65] S. Jha and S. A. Seshia, “A theory of formal synthesis via inductive learning,” *Acta Informatica*, vol. 54, no. 7, pp. 693–726, Nov 2017.

[66] S. Jha, A. Tiwari, and S. Seshia et al., “TeLEx: learning signal temporal logic from positive examples using tightness metric,” *Formal Methods in System Design*, vol. 54, no. 3, pp. 364–387, 2019.

[67] J. L. Morales and J. Nocedal, “Remark on "algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound constrained optimization",” in *ACM Transactions on Mathematical Software*, vol. 38, 2011, pp. 1–4.

[68] M. Eisen, A. Mokhtari, and A. Ribeiro, “A primal-dual quasi-newton method for exact consensus optimization,” *IEEE Transactions on Signal Processing*, vol. 67, no. 23, pp. 5983–5997, 2019.

[69] J. Erway and R. F. Marcia, “Solving limited-memory bfgs systems with generalized diagonal updates,” in *World Congress on Engineering*, 2012.

[70] L. Marcovecchio, “Complications of acute and chronic hyperglycemia,” *US Endocrinology*, vol. 13, no. 1, pp. 17–21, 2017.

[71] K. Vogel, “A comparison of headway and time to collision as safety indicators,” *Accident Analysis Prevention*, vol. 35, no. 3, pp. 427–433, 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0001457502000222

[72] W. Clarke and B. Kovatchev, “Statistical tools to analyze continuous glucose monitor data,” *Diabetes technology & therapeutics*, vol. 11, no. S1, pp. S–45, 2009.

[73] B. P. Kovatchev, “Metrics for glycaemic control - from hba 1c to continuous glucose monitoring,” *Nature Reviews Endocrinology*, vol. 13, no. 7, p. 425, 2017.

[74] B. P. Kovatchev, D. J. Cox, L. A. Gonder-Frederick, D. Young-Hyman, D. Schlundt, and W. Clarke, “Assessment of risk for severe hypoglycemia among adults with iddm: validation of the low blood glucose index.” *Diabetes care*, vol. 21, no. 11, pp. 1870–1875, 1998.

[75] A. H. M. Rubaiyat, Y. Qin, and H. Alemzadeh, “Experimental resilience assessment of an open-source driving agent,” in *2018 IEEE 23rd Pacific rim international symposium on dependable computing (PRDC)*. IEEE, 2018, pp. 54–63.

[76] M. Shane, “Autonomous vehicle radar perception in 360 degrees,” *Nvidia developer blog*, Nov 2019.

[77] “Principles of an open artificial pancreas system (openaps),” https://openaps.org/reference-design.

[78] E. Chertok Shacham, H. Kfir, N. Schwartz, and A. Ishay, "Glycemic control with a basal-bolus insulin protocol in hospitalized diabetic patients treated with glucocorticoids: a retrospective cohort," *BMC Endocr Disord*, vol. 18(75), pp. 1472–6823, 2018.

[79] Perceptus. (2017) Glucosym. [Online]. Available: https://github.com/Perceptus/GlucoSym

[80] R. Visentin, C. Dalla Man, B. Kovatchev, and C. Cobelli, "The university of virginia/padova type 1 diabetes simulator matches the glucose traces of a clinical trial," *Diabetes technology & therapeutics*, vol. 16, no. 7, pp. 428–434, 2014.

[81] B. P. Kovatchev, M. Breton, C. D. Man, and C. Cobelli, "In silico preclinical trials: A proof of concept in closed-loop control of type 1 diabetes," *Journal of Diabetes Science and Technology*, vol. 3, no. 1, pp. 44–55, 2009, pMID: 19444330.

[82] R. Hawkins, C. Paterson, C. Picardi, Y. Jia, R. Calinescu, and I. Habli, "Guidance on the assurance of machine learning in autonomous systems (amlas)," *arXiv preprint arXiv:2102.01564*, 2021.

[83] "openpilot," https://github.com/commaai/openpilot.

[84] "Supported Cars by OpenPilot," https://github.com/commaai/openpilot/blob/master/docs/CARS.md.

[85] "comma.ai," https://comma.ai/.

[86] "Unreal Engine," https://www.unrealengine.com/en-US.

[87] W. G. Najm, J. D. Smith, M. Yanagisawa, and John A. Volpe National Transportation Systems Center (U.S.), "Pre-crash scenario typology for crash avoidance research," Tech. Rep. DOT-VNTSC-NHTSA-06-02, Apr. 2007. [Online]. Available: https://rosap.ntl.bts.gov/view/dot/6281

[88] C. Li, A. Raghunathan, and N. K. Jha, "Hijacking an insulin pump: Security attacks and defenses for a diabetes therapy system," in *2011 IEEE 13th International Conference on e-Health Networking, Applications and Services.* IEEE, 2011, pp. 150–156.

[89] C. M. Ramkissoon, B. Aufderheide, B. W. Bequette, and J. Vehí, "A review of safety and hazards associated with the artificial pancreas," *IEEE reviews in biomedical engineering*, vol. 10, pp. 44–62, 2017.

[90] U.S. Food and Drug Administration, "Class 2 device recall minimed 640g insulin infusion pump," https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfres/res.cfm?id=181608, December 02, 2020.

[91] ——. (2022) Maude adverse event report. [Online]. Available: https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfMAUDE/Detail.cfm?MDRFOI__ID=2430675

[92] ——. (2022) Maude adverse event report. [Online]. Available: https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfMAUDE/detail.cfm?mdrfoi__id=6113784&pc=LZG

[93] T. Bonaci, J. Yan, J. Herron, T. Kohno, and H. Chizeck, "Experimental analysis of denial-of-service attacks on teleoperated robotic systems," in *ACM/IEEE 6th International Conference on Cyber-Physical Systems*, 04 2015, pp. 11–20.

[94] T. Bonaci, J. Herron, T. Yusuf, J. Yan, T. Kohno, and H. Chizeck, "To make a robot secure: An experimental analysis of cyber security threats against teleoperated surgical robots," *arXiv Preprint arXiv:1504.04339*, 04 2015.

[95] V. Sadhu, S. Zonouz, and D. Pompili, "On-board deep-learning-based unmanned aerial vehicle fault cause detection and identification," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 5255–5261.

[96] U.S. Food and Drug Administration, "Class 2 device recall medtronic minimed paradigm model mmt723k insulin pump," https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfRES/res.cfm?id=175210, March 16, 2021.

[97] E. Chien, L. OMurchu, and N. Falliere, "W32.duqu: The precursor to the next stuxnet," in *5th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET 12)*, San Jose, CA, 2012.

[98] K. Zetter, "It's insanely easy to hack hospital equipment," *Wired Magazine*, vol. 16, no. 1, pp. 303–336, 2014.

[99] S. Kim, Y. Eun, and K.-J. Park, "Stealthy sensor attack detection and real-time performance recovery for resilient cps," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 11, 2021.

[100] G. Park, C. Lee, H. Shim, Y. Eun, and K. H. Johansson, "Stealthy adversaries against uncertain cyber-physical systems: Threat of robust zero-dynamics attack," *IEEE Transactions on Automatic Control*, vol. 64, no. 12, pp. 4907–4919, 2019.

[101] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *53rd IEEE Conference on Decision and Control*, 2014, pp. 81–87.

[102] S. Kanderian, S. Weinzimer, and G. Voskanyan et al, "Identification of intraday metabolic profiles during closed-loop glucose control in individuals with type 1 diabetes." *Journal of diabetes science and technology*, vol. 3, no. 5, pp. 1047–1057, 2009.

[103] "longitudinal-maneuvers," https://github.com/commaai/openpilot/tree/master/selfdrive/test/longitudinal_maneuvers.

[104] comma ai, "Bringing forward collision warnings to our open source self-driving car," https://comma-ai.medium.com/bringing-forward-collision-warnings-to-our-open-source-self-driving-car-7545b6e398cd, 2018.

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

[106] A. Tharwat, "Classification assessment methods," *Applied Computing and Informatics*, August 2018.

[107] N. Tatbul, T. J. Lee, S. Zdonik, M. Alam, and J. Gottschlich, "Precision and recall for time series," in *Advances in Neural Information Processing Systems*, 2018, pp. 1920–1930.

[108] X. Zhou and A. Del Valle, "Range based confusion matrix for imbalanced time series classification," in *2020 6th Conference on Data Science and Machine Learning Applications (CDMA)*, 2020, pp. 1–6.

[109] "determine-basal," https://github.com/openaps/oref0/blob/master/lib/determine-basal/determine-basal.js.

[110] OpenAPS community. (2017) Understanding insulin on board (iob) calculations. [Online]. Available: https://openaps.readthedocs.io/en/latest/docs/While%20You%20Wait%20For%20Gear/understanding-insulin-on-board-calculations.html

[111] X. Zhou, A. Schmedding, H. Ren, L. Yang, P. Schowitz, E. Smirni, and H. 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)*, 2022, pp. 79–87.

[112] S. A. Brown, B. P. Kovatchev, D. Raghinaru, J. W. Lum, B. A. Buckingham, Y. C. Kudva, L. M. Laffel, C. J. Levy, J. E. Pinsker, R. P. Wadwa *et al.*, "Six-month randomized, multicenter trial of closed-loop control in type 1 diabetes," *New England Journal of Medicine*, vol. 381, no. 18, pp. 1707–1717, 2019.

[113] S. Matthew, "The limitations of machine learning," *Towards Data Science*, 2019.

[114] H. Lars, "Black-box vs. white-box models," https://towardsdatascience.com/machine-learning-interpretability-techniques-662c723454f3, Mar 2019.

[115] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[116] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: Methods, systems and tools," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 303–336, 2014.

[117] X. Zhou, M. Kouzel, and H. Alemzadeh, “Robustness testing of data and knowledge driven anomaly detection in cyber-physical systems,” in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2022, pp. 44–51.

[118] D. M. Lewis, “Do-it-yourself artificial pancreas system and the openaps movement,” *Endocrinology and Metabolism Clinics*, vol. 49, no. 1, pp. 203–213, 2020.

[119] D. Lewis, S. Leibrand, and . O. Community, “Real-world use of open source artificial pancreas systems,” *Journal of diabetes science and technology*, vol. 10, no. 6, pp. 1411–1411, 2016.

[120] J. Sun, Y. Cao, Q. A. Chen, and Z. M. Mao, “Towards robust lidar-based perception in autonomous driving: General black-box adversarial sensor attack and countermeasures,” in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 877–894.

[121] J. V. Deshmukh, A. Donzé, and S. Ghosh et al., “Robust online monitoring of signal temporal logic,” *Formal Methods in System Design*, vol. 51, no. 1, pp. 5–30, 2017.

[122] A. Camacho and S. A. McIlraith, “Learning interpretable models expressed in linear temporal logic,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, 2019, pp. 621–630.

[123] A. Jones, Z. Kong, and C. Belta, “Anomaly detection in cyber-physical systems: A formal methods approach,” in *53rd IEEE Conference on Decision and Control*, 2014, pp. 848–853.

[124] J. Lamp, S. Silvetti, M. Breton, L. Nenzi, and L. Feng, “A logic-based learning approach to explore diabetes patient behaviors,” in *Computational Methods in Systems Biology*, 2019, pp. 188–206.

[125] R. E. Lyons and W. Vanderkulk, “The use of triple-modular redundancy to improve computer reliability,” *IBM journal of research and development*, vol. 6, no. 2, pp. 200–209, 1962.

[126] Y. Zhang, R. Jetley, P. L. Jones, and A. Ray, “Generic safety requirements for developing safe insulin pump software,” *Journal of diabetes science and technology*, vol. 5, no. 6, pp. 1403–1419, 2011.

[127] N. Paul, T. Kohno, and D. C. Klonoff, “A review of the security of insulin pump infusion systems,” *Journal of diabetes science and technology*, vol. 5, no. 6, pp. 1557–1562, 2011.

[128] L. Meneghetti, G. A. Susto, and S. Del Favero, “Detection of insulin pump malfunctioning to improve safety in artificial pancreas using unsupervised algorithms,” *Journal of diabetes science and technology*, vol. 13, no. 6, pp. 1065–1076, 2019.

[129] X. Zhou, M. Kouzel, C. Smith, and H. Alemzadeh, "Knowsafe: Combined knowledge and data driven hazard mitigation in artificial pancreas systems," *arXiv preprint arXiv:2311.07460*, 2023.

[130] D. Phan, J. Yang, M. Clark, R. Grosu, J. Schierman, S. Smolka, and S. Stoller, "A component-based simplex architecture for high-assurance cyber-physical systems," in *2017 17th International Conference on Application of Concurrency to System Design (ACSD)*, 2017, pp. 49–58.

[131] J. Xu, Z. Zhang, T. Friedman, Y. Liang, and G. V. den Broeck, "A semantic loss function for deep learning with symbolic knowledge," 2018.

[132] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[133] J. Xie. (2018) Simglucose v0.2.1. [Online]. Available: https://github.com/jxx123/simglucose

[134] JDRF/NIDDK Artificial Pancreas Project Consortium, "Reduction of Nocturnal Hypoglycemia by Using Predictive Algorithms and Pump Suspension: An Outpatient Pilot Feasibility and Efficacy Study (PSO3)," https://public.jaeb.org/jdrfapp2/stdy/458.

[135] S. Dutta, T. Kushner, and S. Sankaranarayanan, "Robust data-driven control of artificial pancreas systems using neural networks," in *Computational Methods in Systems Biology*, M. Češka and D. Šafránek, Eds. Cham: Springer International Publishing, 2018, pp. 183–202.

[136] Google Brain team, "TensorFlow Lite for Microcontrollers." 2019. [Online]. Available: https://www.tensorflow.org/lite/microcontrollers

[137] H. Fischer, *A history of the central limit theorem: From classical to modern probability theory.* Springer, 2011.

[138] G. R. Mode and K. A. Hoque, "Adversarial Examples in Deep Learning for Multivariate Time Series Regression," in *49th Annual IEEE Applied Imagery Pattern Recognition (AIPR) workshop 2020*, Sep. 2020. [Online]. Available: http://arxiv.org/abs/2009.11911

[139] Neha Rastogi, "Microchip unveils industry's first MCU with integrated 2d gpu and ddr2 memory." 2019. [Online]. Available: https://www.engineersgarage.com/microchip-unveils-industrys-first-mcu-with-integrated-2d-gpu-and-ddr2-memory/

[140] STMicroelectronics, "ISM330DHCX: machine learning core." 2019. [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/application_note/group1/60/c8/a2/6b/35/ab/49/6a/DM00651838/files/DM00651838.pdf/jcr:content/translations/en.DM00651838.pdf

[141] C. Gupta, A. S. Suggala, A. Goyal, H. V. Simhadri, B. Paranjape, A. Kumar, S. Goyal, R. Udupa, M. Varma, and P. Jain, "Protonn: Compressed and accurate knn for resource-scarce devices," in *International conference on machine learning*. PMLR, 2017, pp. 1331–1340.

[142] Rhonda Dirvin, "Next-generation armv8.1-m architecture: Delivering enhanced machine learning and signal processing for the smallest embedded devices." 2019. [Online]. Available: https://www.arm.com/company/news/2019/02/next-generation-armv8-1-m-architecture

[143] H. Weng, C. Hettiarachchi, C. Nolan, H. Suominen, and A. Lenskiy, "Ensuring security of artificial pancreas device system using homomorphic encryption," *Biomedical Signal Processing and Control*, vol. 79, p. 104044, 2023.

[144] M. R. Aliabadi, M. Seltzer, M. V. Asl, and R. Ghavamizadeh, "Artinali#: An efficient intrusion detection technique for resource-constrained cyber-physical systems," *International Journal of Critical Infrastructure Protection*, vol. 33, p. 100430, 2021.

[145] P. V. Astillo, J. Jeong, W.-C. Chien, B. Kim, J. Jang, and I. You, "Smdaps: A specification-based misbehavior detection system for implantable devices in artificial pancreas system," *Journal of Internet Technology*, vol. 22, no. 1, pp. 1–11, 2021.

[146] O. Vega-Hernandez, F. Campos-Cornejo, D. Campos-Delgado, and D. Espinoza-Trejo, "Increasing security in an artificial pancreas: diagnosis of actuator faults," in *2009 Pan American Health Care Exchanges*. IEEE, 2009, pp. 137–142.

[147] L. Von Rueden, S. Mayer, K. Beckh, B. Georgiev, S. Giesselbach, R. Heese, B. Kirsch, J. Pfrommer, A. Pick, R. Ramamurthy *et al.*, "Informed machine learning–a taxonomy and survey of integrating prior knowledge into learning systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 1, pp. 614–633, 2021.

[148] Y. Liang and G. Van den Broeck, "Learning logistic circuits," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 4277–4286.

[149] B. Chen, Z. Hao, X. Cai, R. Cai, W. Wen, J. Zhu, and G. Xie, "Embedding logic rules into recurrent neural networks," *IEEE Access*, vol. 7, pp. 14 938–14 946, 2019.

[150] S. Guo, Q. Wang, L. Wang, B. Wang, and L. Guo, "Jointly embedding knowledge graphs and logical rules," in *Proceedings of the 2016 conference on empirical methods in natural language processing*, 2016, pp. 192–202.

[151] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[152] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, "Policy distillation," *arXiv preprint arXiv:1511.06295*, 2016.

[153] M. Ma, J. Gao, L. Feng, and J. Stankovic, "Stlnet: Signal temporal logic enforced multivariate recurrent neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 14 604–14 614, 2020.

[154] M. Ma, J. Stankovic, E. Bartocci, and L. Feng, "Predictive monitoring with logic-calibrated uncertainty for cyber-physical systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–25, 2021.

[155] X. Zhou, A. Chen, M. Kouzel, H. Ren, M. McCarty, C. Nita-Rotaru, and H. Alemzadeh, "Runtime stealthy perception attacks against dnn-based adaptive cruise control systems," *arXiv preprint arXiv:2307.08939*, 2024.

[156] B. M. Atchison and P. A. Lindsay, "Safety validation of embedded control software using z animation," in *Proceedings. Fifth IEEE International Symposium on High Assurance Systems Engineering (HASE 2000).* IEEE, 2000, pp. 228–237.

[157] S. Jha, S. Cui, S. Banerjee, J. Cyriac, T. Tsai, Z. Kalbarczyk, and R. K. Iyer, "Ml-driven malware that targets av safety," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN).* IEEE, 2020, pp. 113–124.

[158] R. Krajewski, T. Moers, D. Nerger, and L. Eckstein, "Data-driven maneuver modeling using generative adversarial networks and variational autoencoders for safety validation of highly automated vehicles," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC).* IEEE, 2018, pp. 2383–2390.

[159] M. Fisher, R. C. Cardoso, E. C. Collins, C. Dadswell, L. A. Dennis, C. Dixon, M. Farrell, A. Ferrando, X. Huang, M. Jump *et al.*, "An overview of verification and validation challenges for inspection robots," *Robotics*, vol. 10, no. 2, p. 67, 2021.

[160] "SAE Levels of Driving Automation™ Refined for Clarity and International Audience," https://www.sae.org/blog/sae-j3016-update, 2021.

[161] "Number of autonomous vehicles globally in 2022," 2022. [Online]. Available: https://www.statista.com/statistics/1230664/projected-number-autonomous-cars-worldwide/

[162] "Apollo," https://developer.apollo.auto/.

[163] G. Bishop, G. Welch *et al.*, "An introduction to the kalman filter," *Proc of SIGGRAPH, Course*, vol. 8, no. 27599-23175, p. 41, 2001.

[164] "Adas cameras: How they work and why they need calibration," https://caradas.com/adas-cameras/.

[165] "OpenPilot - Safety Architecture," 2018. [Online]. Available: https://blog.comma.ai/how-to-write-a-car-port-for-openpilot/#background--safety-architecture

[166] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 1625–1634.

[167] C. Ma, N. Wang, Q. A. Chen, and C. 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.

[168] Tencent, "Experimental security research of tesla autopilot," *Tencent Keen Security Lab*, 2019.

[169] Y. Guo, T. Sato, Y. Cao, Q. A. Chen, and Y. Cheng, "Adversarial attacks on adaptive cruise control systems," in *Proceedings of Cyber-Physical Systems and IoT Week 2023*, 2023, pp. 49–54.

[170] T. Sato, J. Shen, N. Wang, Y. Jia, X. Lin, and Q. A. Chen, "Dirty road can attack: Security of deep learning based automated lane centering under {Physical-World} attack," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 3309–3326.

[171] Y. Jia, Y. Lu, J. Shen, Q. A. Chen, Z. Zhong, and T. Wei, "Fooling detection alone is not enough: First adversarial attack against multiple object tracking," in *International Conference on Learning Representations (ICLR)*, 2020.

[172] Y. Ma, J. A. Sharp, R. Wang, E. Fernandes, and X. Zhu, "Sequential attacks on kalman filter-based forward collision warning systems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 10, 2021, pp. 8865–8873.

[173] N. Gavin, K. David S., and G. Steve, "California commercial driver handbook," 2021. [Online]. Available: https://www.dmv.ca.gov/web/eng_pdf/comlhdbk.pdf

[174] L. Chen, T. Tang, Z. Cai, Y. Li, P. Wu, H. Li, J. Shi, J. Yan, and Y. Qiao, "Level 2 autonomous driving on a single device: Diving into the devils of openpilot," *arXiv:2206.08176*, 2022.

[175] Consumer Reports, "CR Active Driving Assistance Systems: Test Results & Design Recommendations," https://data.consumerreports.org/reports/cr-active-driving-assistance-systems/.

[176] A. Schmedding, P. Schowitz, X. Zhou, Y. Lu, L. Yang, H. Alemzadeh, and E. Smirni, "Strategic resilience evaluation of neural networks within autonomous vehicle software," in *43rd International Conference on Computer Safety, Reliability and Security (SafeComp)*, 2024.

[177] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[178] "Cybersecurity risks for hi-tech autonomous and electric vehicles industry," https://www.linkedin.com/pulse/cybersecurity-risks-hi-tech-autonomous-electric-vehicles-samrat-seal/.

[179] H. Wen, Q. A. Chen, and Z. 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)*, 2020, pp. 949–965.

[180] R. Mocnik, D. S. Fowler, and C. Maple, "Vehicular Over-the-Air Software Upgrade Threat Modelling," in *Cenex-LCV and Cenex-CAM 2023*. [Online]. Available: https://wrap.warwick.ac.uk/179188/1/WRAP-vehicular-over-the-air-software-upgrade-threat-modelling-2023.pdf

[181] A. A. Elkhail, R. U. D. Refat, R. Habre, A. Hafeez, A. Bacha, and H. Malik, "Vehicle security: A survey of security issues and vulnerabilities, malware attacks and defenses," *IEEE Access*, vol. 9, pp. 162 401–162 437, 2021.

[182] "Openpilot ssh key security bypass," https://www.redpacketsecurity.com/openpilot-ssh-key-security-bypass/, 2021.

[183] S. Nie, L. Liu, and Y. Du, "Free-fall: Hacking tesla from wireless to can bus," *Briefing, Black Hat USA*, vol. 25, pp. 1–16, 2017.

[184] B. Luo, M. Mosbah, F. Cuppens, L. B. Othmane, N. Cuppens, and S. Kallel, *Risks and Security of Internet and Systems: 16th International Conference, CRiSIS 2021,*. Springer Nature, 2022, vol. 13204.

[185] A. Greenberg, "Hackers remotely kill a jeep on the highway—with me in it," *Wired*, 2015.

[186] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *2010 IEEE symposium on security and privacy*. IEEE, 2010, pp. 447–462.

[187] "comma connect," https://www.comma.ai/connect.

[188] "Installing a fork of openpilot with workbench," https://medium.com/@jfrux/installing-a-fork-of-openpilot-with-workbench-de35e9388021.

[189] M. H. Eiza and Q. Ni, "Driving with sharks: Rethinking connected vehicles with vehicle cybersecurity," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 45–51, 2017.

[190] S. Lagraa, M. Cailac, S. Rivera, F. Beck, and R. State, "Real-time attack detection on robot cameras: A self-driving car application," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2019, pp. 102–109.

[191] D. Rezvani. Hacking automotive ethernet cameras. [Online]. Available: https://argus-sec.com/hacking-automotive-ethernet-cameras/

[192] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA*, vol. 2015, no. S 91, pp. 1–91, 2015.

[193] S. Hoory, T. Shapira, A. Shabtai, and Y. Elovici, "Dynamic adversarial patch for evading object detection models," *arXiv:2010.13070*, 2020.

[194] A. Chahe, C. Wang, A. Jeyapratap, K. Xu, and L. Zhou, "Dynamic adversarial attacks on autonomous driving systems," *arXiv preprint arXiv:2312.06701*, 2023.

[195] G. Lovisotto, H. Turner, I. Sluganovic, M. Strohmeier, and I. Martinovic, "SLAP: Improving physical adversarial examples with short-lived adversarial perturbations," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1865–1882.

[196] Y. Man, R. Muller, M. Li, Z. B. Celik, and R. Gerdes, "That person moves like a car: Misclassification attack detection for autonomous systems using spatiotemporal consistency," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 6929–6946.

[197] "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.

[198] M. Moradi, B. J. Oakes, M. Saraoglu, A. Morozov, K. Janschek, and J. 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)*. IEEE, 2020, pp. 102–109.

[199] H. Alemzadeh, D. Chen, A. Lewis, Z. Kalbarczyk, J. Raman, N. Leveson, and R. Iyer, "Systems-theoretic safety assessment of robotic telesurgical systems," in *Computer Safety, Reliability, and Security: 34th International Conference, SAFECOMP 2015, Delft, The Netherlands, September 23-25, 2015, Proceedings 34*. Springer, 2015, pp. 213–227.

[200] N. Neves, J. Antunes, M. Correia, P. Verissimo, and R. Neves, "Using attack injection to discover new vulnerabilities," in *International Conference on Dependable Systems and Networks (DSN'06)*, 2006, pp. 457–466.

[201] Comma.ai, "Cereal." [Online]. Available: https://github.com/commaai/cereal

[202] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[203] Comma.ai, "log.capnp." [Online]. Available: https://github.com/commaai/cereal/log.capnp

[204] autopi, “CAN DBC file explained.” [Online]. Available: https://www.autopi.io/blog/the-meaning-of-dbc-file/

[205] commaai, “Opendbc.” [Online]. Available: https://github.com/commaai/opendbc

[206] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[207] M. Sundararajan, A. Taly, and Q. Yan, “Axiomatic attribution for deep networks,” in *International conference on machine learning*, 2017, pp. 3319–3328.

[208] R. Schram, A. Williams, and M. van Ratingen, “Implementation of autonomous emergency braking (aeb), the next step in euro ncap’s safety assessment,” *ESV, Seoul*, 2013.

[209] “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, pp. 66–89, 2020.

[210] “GRVA-12-50r1e.pdf,” https://unece.org/sites/default/files/2022-01/GRVA-12-50r1e.pdf.

[211] T. Alsuwian, R. B. Saeed, and A. A. Amin, “Autonomous Vehicle with Emergency Braking Algorithm Based on Multi-Sensor Fusion and Super Twisting Speed Controller,” *Applied Sciences*, vol. 12, no. 17, p. 8458, Aug. 2022.

[212] E. 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.

[213] Comma.ai, “Panda.” [Online]. Available: https://github.com/commaai/panda

[214] K.-T. Cho and K. G. Shin, “Fingerprinting electronic control units for vehicle intrusion detection,” in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 911–927.

[215] F. Croce and M. Hein, “Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks,” in *International conference on machine learning*. PMLR, 2020, pp. 2206–2216.

[216] Wikipedia, “Norm,” https://en.wikipedia.org/wiki/Norm_(mathematics).

[217] I. Urazghildiiev, R. Ragnarsson, P. Ridderstrom, A. Rydberg, E. Ojefors, K. Wallin, P. Enochsson, M. Ericson, and G. Lofqvist, “Vehicle classification based on the radar measurement of height profiles,” *IEEE Transactions on intelligent transportation systems*, vol. 8, no. 2, pp. 245–253, 2007.

[218] H. Schafer, E. Santana, A. Haden, and R. Biasini, “A commute in data: The comma2k19 dataset,” *arXiv:1812.05752*, 2018.

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

[220] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, “Towards the science of security and privacy in machine learning,” *arXiv:1611.03814*, 2016.

[221] A. Kurakin, I. J. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” in *Artificial intelligence safety and security.* Chapman and Hall/CRC, 2018, pp. 99–112.

[222] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” *arXiv:1706.06083*, 2017.

[223] K. Xu, Y. Xiao, Z. Zheng, K. Cai, and R. Nevatia, “PatchZero: Defending against Adversarial Patch Attacks by Detecting and Zeroing the Patch,” in *IEEE/CVF Winter Conference on Applications of Computer Vision*, Jan. 2023, pp. 4621–4630.

[224] Z. Chen, P. Dash, and K. 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.* ACM, Jul. 2023, pp. 689–703.

[225] J. Liu, A. Levine, C. P. Lau, R. Chellappa, and S. 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).* IEEE, Jun. 2022, pp. 14 953–14 962.

[226] C. Xiang, A. N. Bhagoji, V. Sehwag, and P. Mittal, “Patchguard: A provably robust defense against adversarial patches via small receptive fields and masking,” in *30th USENIX Security Symposium*, 2021, pp. 2237–2254.

[227] Y. Zhang and P. Liang, “Defending against whitebox adversarial attacks via randomized discretization,” in *The 22nd International Conference on Artificial Intelligence and Statistics*, 2019, pp. 684–693.

[228] G. K. Dziugaite, Z. Ghahramani, and D. M. Roy, “A study of the effect of jpg compression on adversarial images,” *arXiv:1608.00853*, 2016.

[229] W. Xu, D. Evans, and Y. Qi, “Feature squeezing: Detecting adversarial examples in deep neural networks,” in *Network and Distributed Systems Security Symposium (NDSS) 2018*, 2017.

[230] R. S. Hallyburton, Y. Liu, Y. Cao, Z. M. Mao, and M. Pajic, “Security analysis of camera-lidar fusion against black-box attacks on autonomous vehicles,” in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 1903–1920.

[231] X. Liu, H. Yang, Z. Liu, L. Song, H. Li, and Y. Chen, "Dpatch: An adversarial patch attack on object detectors," *arXiv:1806.02299*, 2018.

[232] M. Lee and Z. Kolter, "On physical adversarial patches for object detection," *arXiv:1906.11897*, 2019.

[233] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *IEEE conference on computer vision and pattern recognition*, 2016, pp. 2574–2582.

[234] Z. Wu, S.-N. Lim, L. S. Davis, and T. Goldstein, "Making an invisibility cloak: Real world adversarial attacks on object detectors," in *Computer Vision-ECCV*, 2020, pp. 1–17.

[235] T. Sato, Y. Hayakawa, R. Suzuki, Y. Shiiki, K. Yoshioka, and Q. A. Chen, "WIP: Practical Removal Attacks on LiDAR-based Object Detection in Autonomous Driving," in *Inaugural International Symposium on Vehicle Security & Privacy*, 2023.

[236] J. Shen, J. Y. Won, Z. Chen, and Q. A. 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*, 2020, pp. 931–948.

[237] R. Komissarov and A. Wool, "Spoofing attacks against vehicular fmcw radar," in *Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security*, 2021, pp. 91–97.

[238] T. Sato, S. H. V. Bhupathiraju, M. Clifford, T. Sugawara, Q. A. Chen, and S. Rampazzi, "WIP: Infrared Laser Reflection Attack Against Traffic Sign Recognition Systems," in *Proceedings Inaugural International Symposium on Vehicle Security & Privacy*, 2023.

[239] P. Sun, L. Garcia, and S. Zonouz, "Tell me more than just assembly! reversing cyber-physical execution semantics of embedded iot controller software binaries," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 349–361.

[240] O. Gheibi, D. Weyns, and F. Quin, "Applying machine learning in self-adaptive systems: A systematic literature review," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 15, no. 3, pp. 1–37, 2021.

[241] S. M. Hezavehi, D. Weyns, P. Avgeriou, R. Calinescu, R. Mirandola, and D. Perez-Palacin, "Uncertainty in self-adaptive systems: A research community perspective," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 15, no. 4, pp. 1–36, 2021.

[242] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," *arXiv preprint arXiv:1610.02136*, 2016.

[243] R. Kaur, K. Sridhar, S. Park, S. Jha, A. Roy, O. Sokolsky, and I. Lee, “Codit: Conformal out-of-distribution detection in time-series data,” *arXiv preprint arXiv:2207.11769*, 2022.

[244] G. Bombara and C. Belta, “Offline and online learning of signal temporal logic formulae using decision trees,” *ACM Transactions on Cyber-Physical Systems*, vol. 5, no. 3, pp. 1–23, 2021.

[245] V. Singh and A. Thurman, “How many ways can we define online learning? a systematic literature review of definitions of online learning (1988-2018),” *American Journal of Distance Education*, vol. 33, no. 4, pp. 289–306, 2019.

[246] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas *et al.*, “Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav),” in *International conference on machine learning.* PMLR, 2018, pp. 2668–2677.

[247] K. Rasheed, A. Qayyum, M. Ghaly, A. Al-Fuqaha, A. Razi, and J. Qadir, “Explainable, trustworthy, and ethical machine learning for healthcare: A survey,” *Computers in Biology and Medicine*, vol. 149, p. 106043, 2022.

[248] J. He, S. L. Baxter, J. Xu, J. Xu, X. Zhou, and K. Zhang, “The practical implementation of artificial intelligence technologies in medicine,” *Nature Medicine*, vol. 25, no. 1, pp. 30–36, 2019.

[249] R. Challen, J. Denny, M. Pitt, L. Gompels, T. Edwards, and K. Tsaneva-Atanasova, “Artificial intelligence, bias and clinical safety,” vol. 28, no. 3, pp. 231–237, 2019.

[250] (2018) ISO 26262: Road vehicles — functional safety.

[251] (2019) ISO/PAS 21448: Road vehicles — safety of the intended functionality.

[252] U. Food, D. Administration *et al.*, “The content of investigational device exemption (ide) and premarket approval (pma) applications for artificial pancreas device systems,” *Silver Spring*, 2012.

[253] S. Burton, I. Kurzidem, A. Schwaiger, P. Schleiss, M. Unterreiner, T. Graeber, and P. Becker, “Safety assurance of machine learning for chassis control functions,” in *Computer Safety, Reliability, and Security*, I. Habli, M. Sujan, and F. Bitsch, Eds. Cham: Springer International Publishing, 2021, pp. 149–162.

[254] L. Gauerhof, R. Hawkins, C. Picardi, C. Paterson, Y. Hagiwara, and I. Habli, “Assuring the safety of machine learning for pedestrian detection at crossings,” in *Computer Safety, Reliability, and Security*, A. Casimiro, F. Ortmeier, F. Bitsch, and P. Ferreira, Eds. Cham: Springer International Publishing, 2020, pp. 197–212.

[255] S. Burton, L. Gauerhof, B. B. Sethy, I. Habli, and R. Hawkins, “Confidence arguments for evidence of performance in machine learning for highly automated driving functions,” in *Computer Safety, Reliability, and Security*, 2019.

[256] C. Picardi, R. Hawkins, C. Paterson, and I. Habli, "A pattern for arguing the assurance of machine learning in medical diagnosis systems," in *Computer Safety, Reliability, and Security*, A. Romanovsky, E. Troubitsyna, and F. Bitsch, Eds. Springer International Publishing, 2019, pp. 165–179.

[257] H. Thabit and R. Hovorka, "Coming of age: the artificial pancreas for type 1 diabetes," *Diabetologia*, vol. 59, no. 9, pp. 1795–1805, 2016.

[258] S. Kapil, R. Saini, S. Wangnoo, and S. Dhir, "Artificial pancreas system for type 1 diabetes—challenges and advancements," *Exploratory Research and Hypothesis in Medicine*, vol. 5, no. 3, pp. 110–120, 2020.

[259] M. Bagheri, J. Lamp, X. Zhou, and H. Alemzadeh, "Towards developing safety assurance cases for learning-enabled medical cyber-physical systems," in *SafeAI: The AAAI's Workshop on Artificial Intelligence Safety*, 2023.

[260] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 11 264–11 272.

[261] Tandem Diabet Care. (2022) Basal-IQ. [Online]. Available: https://www.tandemdiabetes.com/providers/products/basal-iq

[262] M. D. Breton, L. G. Kanapka, R. W. Beck, L. Ekhlaspour, G. P. Forlenza, E. Cengiz, M. Schoelwer, K. J. Ruedy, E. Jost, L. Carria, E. Emory, L. J. Hsu, M. Oliveri, C. C. Kollman, B. B. Dokken, S. A. Weinzimer, M. D. DeBoer, B. A. Buckingham, D. Cherñavvsky, R. P. Wadwa, and iDCL Trial Research Group, "A Randomized Trial of Closed-Loop Control in Children with Type 1 Diabetes," *The New England Journal of Medicine*, vol. 383, no. 9, pp. 836–845, Aug. 2020.

[263] D. M. Maahs, P. Calhoun, B. A. Buckingham, H. P. Chase, I. Hramiak, J. Lum, F. Cameron, B. W. Bequette, T. Aye, T. Paul, R. Slover, R. P. Wadwa, D. M. Wilson, C. Kollman, R. W. Beck, and In Home Closed Loop Study Group, "A randomized trial of a home system to reduce nocturnal hypoglycemia in type 1 diabetes," *Diabetes Care*, vol. 37, no. 7, pp. 1885–1891, Jul. 2014.

[264] "Nvidia drive sim," https://www.nvidia.com/en-us/self-driving-cars/simulation/.

[265] "Autonomous driving starts to hit mainstream as 3.5 million new cars had level 2 features in q4 2020?" https://canalys-prod-public.s3.eu-west-1.amazonaws.com/static/press_release/2021/CanalysAutomediaalertQ420L2WW.pdf, 2021.

[266] T. Pritchard, "Self-driving cars: Here's what the autonomous driving levels mean," https://www.tomsguide.com/reference/self-driving-cars-heres-what-the-autonomous-driving-levels-mean.

[267] P. Insel, K. Kramer, R. Sherwin, J. Liljenquist, J. Tobin, R. Andres, and M. Berman, "Modeling the insulin-glucose system in man," *Fed Proc*, vol. 33, no. 7, pp. 1865–1868, Jul 1974.

[268] R. N. Bergman, D. T. Finegood, and M. Ader, "Assessment of insulin sensitivity in vivo," *Endocrine reviews*, vol. 6, no. 1, pp. 45–86, 1985.

[269] R. S. Sherwin, K. J. Kramer, J. D. Tobin, P. A. Insel, J. E. Liljenquist, M. Berman, R. Andres *et al.*, "A model of the kinetics of insulin in man," *The Journal of clinical investigation*, vol. 53, no. 5, pp. 1481–1492, 1974.

[270] A. Melmer, T. Züger, D. M. Lewis, S. Leibrand, C. Stettler, and M. Laimer, "Glycaemic control in individuals with type 1 diabetes using an open source artificial pancreas system (openaps)," *Diabetes, Obesity and Metabolism*, vol. 21, no. 10, pp. 2333–2337, 2019.

[271] O. community. (2017) Understanding the determine-basal logic. [Online]. Available: https://openaps.readthedocs.io/en/latest/

[272] J. Walsh, R. Roberts, T. S. Bailey, and L. Heinemann, "Bolus advisors: Sources of error, targets for improvement," *Journal of Diabetes Science and Technology*, vol. 12, no. 1, pp. 190–198, 2018.

[273] F. Cameron, B. W. Bequette, D. M. Wilson, B. A. Buckingham, H. Lee, and G. Niemeyer, "A Closed-Loop Artificial Pancreas Based on Risk Management," *Journal of Diabetes Science and Technology*, vol. 5, no. 2, pp. 368–379, Mar. 2011. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3125931/

[274] P. Viroonluecha, E. Egea-Lopez, and J. Santa, "Evaluation of blood glucose level control in type 1 diabetic patients using deep reinforcement learning," *Plos one*, vol. 17, no. 9, p. e0274608, 2022.

[275] G. Fico, L. Hernández, J. Cancela, M. M. Isabel, A. Facchinetti, C. Fabris, R. Gabriel, C. Cobelli, and M. T. Arredondo Waldmeyer, "Exploring the frequency domain of continuous glucose monitoring signals to improve characterization of glucose variability and of diabetic profiles," *Journal of diabetes science and technology*, vol. 11, no. 4, pp. 773–779, 2017.

[276] A. B. King and D. U. Armstrong, "A Prospective Evaluation of Insulin Dosing Recommendations in Patients with Type 1 Diabetes at Near Normal Glucose Control: Bolus Dosing," *Journal of diabetes science and technology (Online)*, vol. 1, no. 1, pp. 42–46, Jan. 2007. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2769601/

[277] U. Food and D. Administration. (2022) Cfr - code of federal regulations title 21. [Online]. Available: https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/cfrsearch.cfm?fr=312.32

[278] ——. (2020) Recalls, corrections and removals (devices). [Online]. Available: https://www.fda.gov/medical-devices/postmarket-requirements-devices/recalls-corrections-and-removals-devices

[279] ——. (2022) Maude adverse event report. [Online]. Available: https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfMAUDE/Detail.cfm?MDRFOI__ID=2417117

[280] H. Alemzadeh, D. Chen, Z. Kalbarczyk, R. K. Iyer, X. Li, T. Kesavadas, and J. Raman, "A software framework for simulation of safety hazards in robotic surgical systems," in *Special Issue on Medical Cyber Physical Systems Workshop*, vol. 12, no. 4. SIGBED Review, 2015.

[281] Y. Xu, D. Tran, Y. Tian, and H. Alemzadeh, "Analysis of cyber-security vulnerabilities of interconnected medical devices," in *2019 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, 2019.

[282] J. G. Gaspar and D. V. McGehee, "Driver brake response to sudden unintended acceleration while parking," *Transportation Research Interdisciplinary Perspectives*, vol. 2, p. 100039, 2019.

[283] B. Weng, M. Zhu, and K. Redmill, "A formal safety characterization of advanced driver assist systems in the car-following regime with scenario-sampling," *IFAC-PapersOnLine*, vol. 55 no.24, pp. 266–272, 2022.

[284] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: why and how you should (still) use dbscan," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 3, pp. 1–21, 2017.

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

[286] "Qualtrics," https://www.qualtrics.com/.